N° 75 mensuel Janvier-Février 1992-24 F

# La Revue de l'Utilisateur PC

*Le magazine professionnel en programmation PC*

**RECONSTRUCTION d'une File Allocation Table**

**PDEL un utilitaire qui vous permet de protéger la confidentialité de certaines données en les détruisant de façon définitive**

**LA PROGRAMMATION ORIENTEE OBJET en PASCAL**

**WINDOWS 3.0 dans le contexte des années 90**

**NOTION D'ALGORITHMIQUE Les structures de données évoluées**

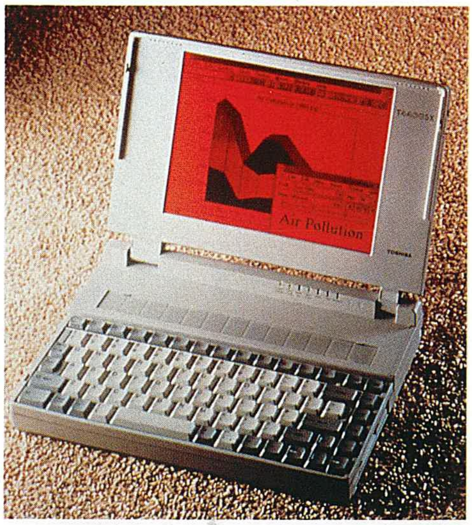
**LA LOGIQUE A 3 ENTREES**

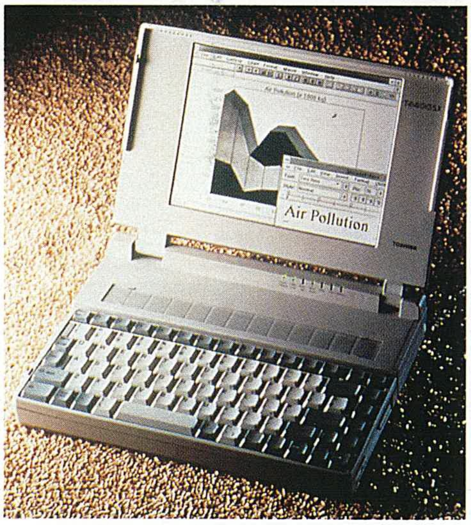
JANVIER-FÉVRIER 1992 - N° 75 - 24 F - Suisse 8 FS - Belgique 200 FB - Canada 6,25 $ Can

M 2537 - 75 - 24,00 F -RD

# LA DÉDICACE INFORMATIQUE VOUS PRÉSENTE LES DERNIERS-NÉS DE TOSHIBA EN DÉTAIL

Les nouveaux notebooks Toshiba profitent des derniers progrès réalisés en matière de miniaturisation : format A4, coque en fibre de carbone, lecteur de disquettes ultra-plat 720 ko et 1,44 Mo, batteries nouvelle génération, etc. Enfin, ils sont extractibles, c'est-à-dire qu'ils s'encastrent dans la Desk Station IV aussi facilement qu'un autoradio. Celle-ci transforme les notebooks en micros de bureau complet avec écran couleur, clavier 102 touches et deux emplacements pour cartes longues.




Deux versions du T4400SX sont disponibles: une avec un écran VGA plasma et une avec un écran VGA LCD.

**▲ Le T4400 SX**

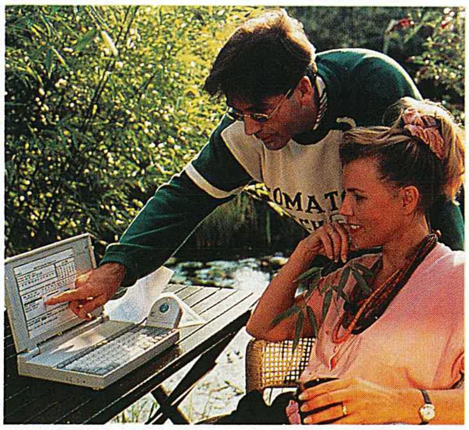
- Intel486 SX à 25 MHz, upgradable en Intel486 DX.
- 8 ko de mémoire cache.
- De 2 à 10 Mo de RAM.
- Ecran VGA plasma ou LCD.
- Disque dur de 80 Mo.
- 3,3 kg.

**Compouce Quadri**

- Modem externe de la taille d'un briquet (poids : 90 g).
- Permet l'accès aux services vidéotex (Minitel, messageries d'entreprises, etc.), le transfert de fichiers (2400 bit/s) et l'émulation des sites centraux et des serveurs utilisant la norme VT100.
- Compatible Hayes et donc avec la plupart des logiciels du marché.

**Le T3300 SL**

- Intel386 SL à 25 MHz.
- 64 ko de mémoire cache.
- De 2 à 10 Mo de RAM.
- Ecran LCD VGA.
- Disque dur de 80 Mo.
- Emplacement 16 bits acceptant les cartes d'extension au format cartes de crédit (modem, réseau, mémoires flash, etc.) au standard PCMCIA 2.0.
- 2,7 kg.
- Disponible fin janvier 1992. ▼



Des notebooks de la taille d'une feuille A4!

Pratique! Les nouveaux notebooks Toshiba s'encastrent facilement dans la Desk Station IV.

**▲ Le T2200 SX**

- Intel386 SX à 20 MHz.
- De 2 à 10 Mo de RAM.
- Ecran LCD VGA.
- Disque dur de 40 ou 60 Mo.
- Emplacement dédié modem au format carte de crédit.
- 2,5 kg.

**L'ExpressWriter 201**

- La plus petite imprimante portative du monde (303 x 80 x 44 mm), compatible avec l'ensemble des logiciels du marché.
- Ultra-légère, 850 g sans la batterie.
- Autonome (une heure avec sa batterie).
- Possibilité d'imprimer sur papier thermique, sur papier standard, transparent ou même sur des enveloppes.
- Deux polices résidentes : Courier et Prestige Elite.

# La Dédicace INFORMATIQUE

173, rue de Vaugirard 75015 Paris. Tél : 45 66 94 69. Fax : 43 06 35 18. Télex : 202 993

SOMMAIRE


# 2

## WINDOWS 3.0 DANS LE CONTEXTE DES ANNEES 90

A voir le déferlement de logiciels tournant sous WINDOWS 3.0, il est permis de s'interroger sur le devenir des environnements micro des années 90

Par Daniel VASSEUR

# 6

## NOTION D'ALGORITHMIQUE LES STRUCTURES DE DONNEES EVOLUEES

Dans cette série d'articles, je vous propose de découvrir certaines structures de données évoluées très couramment utilisées en informatique ; nous étudierons le fonctionnement et l'implémentation en langage C des piles, des files, des listes chaînées et des arbres binaires

Par Frédéric BERTON

# 16

## LA PROGRAMMATION ORIENTEE OBJET EN PASCAL

Depuis quelques années, la Programmation Orientée Objet gagne du terrain. Cette nouvelle méthode peut sembler quelque peu compliquée. Il n'en est rien. Comme toute méthode, il suffit de prendre le pli. Nous allons essayer de vous en expliquer toutes les ficelles. Le langage utilisé est le PASCAL (TURBO PASCAL Version 6)

Par François MONTBAZET

# 28

## LA LOGIQUE A 3 ENTREES

La logique constitue une des disciplines de base de l'informatique : Algèbre de Boole, instructions logiques des microprocesseurs, ou fonctions logiques des tableurs et des langages de programmation... témoignent de son caractère à la fois omniprésent et prépondérant

Par Paul FRANCESCHI

# 44

## PDEL : UN UTILITAIRE QUI VOUS PERMET DE PROTEGER LA CONFIDENTIALITE DE CERTAINES DONNEES EN LES DETRUISANT DE FAÇON DEFINITIVE

Le système MS-DOS fait preuve, reconnaissons-le, d'une certaine clémence vis-à-vis de l'utilisateur distrait

Par Jean-Michel SPINER

# 46

## RECONSTRUCTION D'UNE FILE ALLOCATION TABLE

Récupérer un fichier ASCII ou un fichier de traitement de texte sur une disquette dont les Files Allocations Tables ont été effacées. (Peut-être à cause d'un cheval de Troie...) ou d'un formatage interrompu

Par Jacques ROUILLIER



La Revue de l'Utilisateur PC

Le magazine professionnel en programmation PC

Mensuel 24 F
2, rue d'Amsterdam
75009 Paris
Tél. : (1) 42.81.54.27.
Service abonnement :
Tél. : (1) 42.81.54.27.

Directrice de la Publication :
Sardie ERRERA

Editeur :
Roger CHRISTOPHE

Rédacteur en chef :
Daniel VASSEUR

Directeur de fabrication :
Michel MILLET

Directeur de publicité :
Liliane ERRERA
Tél. : (1) 42.67.93.80.

Edité par SOFTin Communications (France) Sarl
R.C. Paris B 329 834 592
N° Siret
329 834 592 000 19
Commission Paritaire (en cours)
Dépôt légal :
1er trimestre 1992

CCP Paris 8850 84 D

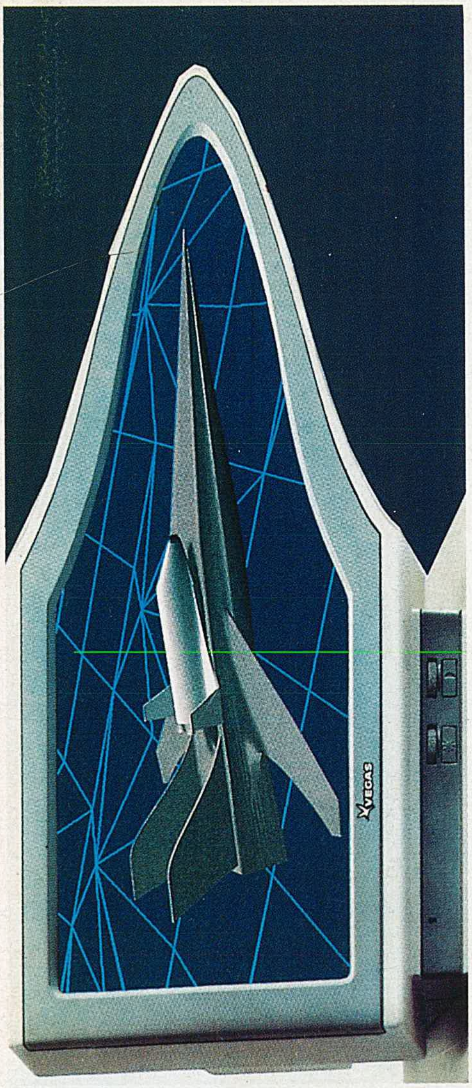
Impression :
SNIL
93600 Aulnay-sous-Bois

Composition :
SOFTin Communications

# WINDOWS 3.0 dans le contexte des années 90

**A voir le déferlement de logiciels tournant sous WINDOWS 3.0, il est permis de s'interroger sur le devenir des environnements micro des années 90.**

Daniel VASSEUR




Aujourd'hui, le monde de la micro-informatique est dominé par la présence des machines compatibles IBM fonctionnant sous DOS.

Le PC a été rendu célèbre autant par son système d'exploitation que par son architecture ouverte qui lui permettait d'évoluer dans le temps sans remettre perpétuellement en cause son existence propre.

Cependant, DOS devenait trop étroit, trop synonyme de machine ancienne aux capacités limitées, trop faible compétiteur de la convivialité légendaire des MACINTOSH.

L'avénement des machines de type 80386 remettait sérieusement en cause l'existant matériel et logiciel : pour la première fois dans le monde des IBM et compatibles, on obtenait des possibilités de développement quasiment illimitées aussi bien en terme d'espace mémoire qu'en matière de multitâches ou encore de puissance.

Les 386 pouvaient gommer d'un coup, d'un seul, toutes les habitudes ancestrales des développeurs, et le dilemme était ainsi posé : cohabiter avec 2 générations de machines dès lors qualifiées de poussives, ou bien changer complètement le fusil d'épaule et décider d'appliquer aux 386 un environnement d'exploitation totalement nouveau et à la hauteur des ambitions humaines.

C'est pour finir une option tierce et beaucoup plus progressive qui a été retenue.

La bonne vieille solution de l'intégrateur est revenue en surface et WINDOWS 3.0, surcouche au DOS, est apparu avec son lot de nouveautés par rapport à ses précédentes versions.

A mon avis, WINDOWS, qui dispose quasiment de la même interface utilisateur que le système d'exploitation OS/2, était vraisemblablement prévu non pour concurrencer ce dernier, mais plus logiquement pour convertir les derniers récalcitrants aux bienfaits d'un système d'exploitation multitâches doté d'une interface graphique dont personne n'aurait à rougir.

Seulement voilà ! : le succès d'OS/2 a été trop intimement lié à celui des PS/2, les constructeurs et les consommateurs - plus souvent qualifiés d'utilisateurs - ont réagi face au coup de force d'IBM qui voulait imposer son nouveau bus MCA. En effet, dans le même temps, d'autres constructeurs proposaient des machines au moins aussi performantes que la première génération de PS/2, mais qui, outre un prix plus intéressant, offraient divers avantages sur leurs concurrentes comme échapper à la pénurie des cartes additionnelles très rares à l'époque sur MCA.

Quand OS/2 est réellement apparu, le marché était alors divisé en deux clans : les tout IBM qui n'avaient pas encore réalisé complètement la migration de leur parc machines et les autres qui avaient résolument adopté les machines compatibles offertes par COMPAQ et autres pour des raisons d'homogénéité de parc.

Les uns et les autres avaient dépensé beaucoup d'argent, mais pour les seconds, il n'y avait aucune arrière-pensée OS/2 car en 1989, les applications courantes se satisfaisaient des 640 KO du DOS et d'un peu de mémoire EMS.

Le choix d'OS/2 imposait un nouvel investissement important tant sur le plan matériel (ajout de mémoire, remplacement des disques durs taillés trop petits,...) que sur celui du logiciel puisqu'il fallait renouveler quasiment toute la logithèque, et les catalogues de l'époque relevaient plus de la déclaration d'intentions que de la réalité quotidienne.

Pour compliquer encore plus, l'offre OS/2 était divisée entre celle d'IBM et celle de MICROSOFT. Entre ces offres concurrentes les décideurs avaient bien du mal à faire leur choix.

Avec 4 années de recul, il est aisé de décrire les événements qui ont fait que le marché n'a pas basculé vers un système d'exploitation nouveau et réellement attractif.

A la date d'aujourd'hui (Janvier 1992), seul IBM soutient encore des développements sur une nouvelle version de ce système d'exploitation qui ne devrait fonctionner que sur machines de type 386. Des BETA TEST - déjà très stabilisées - d'OS/2 2.0 circulent depuis quelques temps chez certains gros clients à la patience d'ange.

Pourtant, une question d'importance vitale ne peut que se poser dans tous les esprits pragmatiques : est-ce que cette nouvelle version ne sera pas "propriétaire" puisque seulement supportée par IBM ?

Autrement dit : pourra-t-on faire fonctionner OS/2 2.0 sur n'importe quelle machine 386, du PS/2 8595 au clone Taïwanais à 20 000 F TTC ?

Dans un passé récent, IBM a déjà cédé à la tentation avec une version propriétaire d'UNIX, baptisée AIX, qui ne pouvait fonctionner que sur les PS/2 et qui, soit dit en passant, étant présentée comme l'alternative à OS/2...

Nous voici donc en 1992 face à un marché morcelé qui, pour le plus grand bien des utilisateurs finaux, n'échappe pas à la tradition des offres concurrentes qui font tant avancer les techniques.

A bien y regarder, l'offre OS/2, même si elle est loin de faire l'unanimité, présente pourtant bien des similitudes au niveau du développement avec ce qui se passe sous WINDOWS : il est couramment dit que les développeurs sont communs aux 2 environnements à hauteur de 80 %.

Des éditeurs de logiciels proposent d'ailleurs les deux versions de leurs produits dans un même package (WINGZ d'INFORMIX SOFTWARE par exemple).

En essayant d'analyser les raisons qui font qu'un décideur choisit WINDOWS plutôt qu'OS/2, je pourrai citer pêle-mêle :

• la sécurité morale (un peu illusoire) de ne pas avoir à changer de système d'exploitation ;

• la garantie que les applications DOS existantes s'exécutent normalement (au prix parfois de quelques prouesses de paramètrage) ;

• les faibles contraintes d'installation puisque, par exemple, il n'y a pas de reformatage du disque ;

• la certitude d'installer WINDOWS sur toutes les générations de machines.

En ce qui concerne ce dernier point, je vous ai déjà donné mon avis au cours des précédents articles consacrés à WINDOWS (cf exemplaires de La Revue de l'Utilisateur PC n° 71 à 74 de notre magazine).

Le temps pour vous de vous procurer ces indispensables lectures, je rappellerai simplement que WINDOWS fonctionne selon 3 modes totalement différents selon le type de la machine hôte et il est évident que le bon vieil XT des familles soigneusement confié à la secrétaire qui ne connaît encore que MULTIPLAN ou VISIO, se montre très insuffisant pour permettre de travailler !

Dans le haut de gamme aussi, il existe des machines qui refusent de faire tourner WINDOWS, ou qui le font de façon incomplète comme par exemple l'IBM 8570-A21.

Je vous ai déjà parlé, WINDOWS ne sait réellement travailler que sur des machines de type 80386 très largement dotées en ressources matérielles.

Ce qui me gêne le plus dans ce constat, c'est de voir sans cesse reculer la limite du normal.

Il y a encore quelques mois, j'utilisais WINDOWS de façon relativement confortable sur un 386SX à 16 MHz.

L'introduction d'images graphiques sous WINWORD commençant à peser sérieusement sur les délais d'affichage, j'ai alors changé de traitement de textes pour passer à AMIPRO 1.2 qui présentait des caractéristiques de rapidité capables de me redonner le sourire.

Quelques temps après, j'ai troqué mon 386SX-16 contre un 386XS-20 pour environ 25 % de puissances supplémentaire, et tout semblait pour le mieux dans le meilleur des mondes.

Les logiciels évoluant, j'ai tout naturellement opté pour la nouvelle version de ce traitement de textes désormais commercialisé par LOTUS.

La version 2.0 d'AMIPRO est superbe, tout est là pour me combler d'aise : cadres, filets, images, dessins, graphiques, vastes possibilités de mise en page,...

Elle est aussi très gourmande et mon SX-20 est désormais trop limité en terme de performances microprocesseur pour que certains traitements restent supportables, et il faut à nouveau avoir recours aux options telles que le non-affichage des images à l'écran pour obtenir des temps de réponses confortables.

Fait significatif, ma configuration mémoire est désormais insuffisante : 4 MO de RAM dont 512 KO réservés à un cache-disque imposent un swap incessant sur disque dès lors que mon traitement de textes se trouve en mémoire.

A titre d'exemple, AMIPRO est fourni avec un module complémentaire qui permet de traiter les impressions en arrière-plan.

Dans le principe, AMIPRO crée la tâche AMIPRINT, lui indique les travaux à effectuer, et redonne immédiatement la main à l'utilisateur pendant que la tâche AMIPRINT se débrouille avec les fichiers à traiter et le Gestionnaire d'Impressions de WINDOWS. Tout cela serait merveilleux si la tâche AMIPRINT ne consommait pas 620 KO de mémoire à elle seule !

La documentation ne nous prend pas en traître : AMIPRO nécessite 2 MO de mémoire vive pour s'exécuter normalement - tout comme LOTUS pour WINDOWS d'ailleurs -, et dans la documentation des produits WINDOWS, il faut toujours comprendre que ce chiffre indique la taille de la mémoire réellement disponible après chargement de WINDOWS.

Le calcul est alors simple à faire :

• 930 KO de mémoire pour la session DOS de base,

• 850 KO de mémoire pour WINDOWS,

• 512 KO réservés pour le cache-disque,

sur une machine de 4 MO, il reste moins que les 2 MO requis par AMIPRO et, mon grand désespoir, les mécanismes de paging du mode 386 étendu se mettent en œuvre.

J'ai fait différents tests, et j'ai constaté que, pour éviter le paging sous AMI-PRO 2.0, il fallait équiper la machine avec au moins 5 MO de mémoire vive. Replacez cette information dans le contexte des années 88-89 et essayez un peu d'imaginer la réaction d'un utilisateur des traitements de textes d'alors !

Heureusement pour mon système nerveux, je dispose d'une autre machine pour travailler : un 80386 à 40 MHz équipé de 8 MO de RAM et d'un coprocesseur arithmétique, ce qui me permet tout à la fois :

- de lancer simultanément plusieurs tâches avant de subir les effets de la pagination,
- d'exécuter AMIPR sans inhiber l'affichage des dessins et graphiques.

Pour WINDOWS 3.0, le bilan est lourd, tout autant d'ailleurs que celui d'OS/2 : il faut des machines qualifiées aujourd'hui de puissantes pour supporter nos exigences dans le domaine du confort.

Sur plate-forme type PC, nous nous rapprochons de plus en plus de la puissance affichée par ces monstres sacrés de la DAO-CAO que sont les stations SUN. Un 486 à 33 MHz développe couramment 16 MIPS, c'est-à-dire environ la moitié de la puissance d'une station SUN haut de gamme.

Avec l'arrivée en force des interfaces graphiques de type WINDOWS sous UNIX, il est possible d'imaginer qu'à échéance de quelques années, l'harmonisation aura fait son œuvre, et que les utilisateurs n'auront plus à se poser la question du système d'exploitation animant la machine et même, de la marque et du type de la machine elle-même.

Que la machine soit équipée d'un microprocesseur MOTOROLA ou INTEL, cette question n'aura alors plus d'importance. (Combien d'entre nous connaissent la marque du carburateur de leur véhicule ?).

Les applications auront le même look, utiliseront les mêmes serveurs de données fonctionnant indifféremment sous UNIX, OS/2, NETWARE ou d'autres encore : les micro-ordinateurs seront totalement banalisés et le consommateur fera son choix sur des critères autres que ceux des techniciens comme par exemple la couleur de la caisse ou la forme de l'écran.

Dans cet univers de visionnaire, un OS/2 devenu crédible pourrait avoir une place de choix : n'oublions pas qu'avant lui, WINDOWS aussi a eu des faux pas !

---


NOUVEAU !

# AG - OUTIL

## Outil de programmation pour le développement en CLIPPER ©

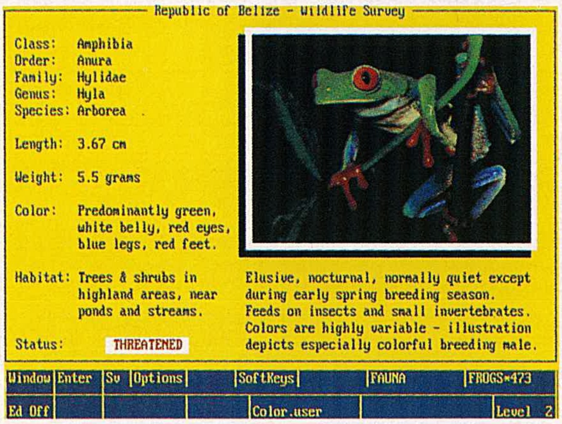
- **Réalisation instantanée:**
  64 types de menus possibles,
  Pages écrans,
  Mise à jour rapide de données sous forme de liste,
  Formats d'impression
- **Macro environnement:**
  Réduit de 50% le temps de programmation
- **Entièrement paramétrable:**
  Souplesse et sur mesure
- **Protection d'accès** (10 niveaux)
- **Multifichiers**

Pour tout renseignement ou documentation appeler le 43.96.93.18 ou fax. 43.96.07.44

CB informatique
30, avenue du Maréchal de Lattre de Tassigny
94410 SAINT-MAURICE
Tél. 43.96.93.18 - Fax : 43.96.07.44

CB informatique
NORD PAS DE CALAIS
Tél. 21.01.09.12

**SUMMER WINDOWS Frs 4 900 HT**
Un environnement de programmation graphique
Gestion d'Interface Graphique
Dépassement des limites de taille du DOS




**DGT Frs 3 250 HT**
pour incorporer images,
photos dans vos champs mémos

~~10.500 FHT~~
**5900 Frs H.T.**
**NANTUCKET 5.01**
**Version française**
+ Port 100 Frs

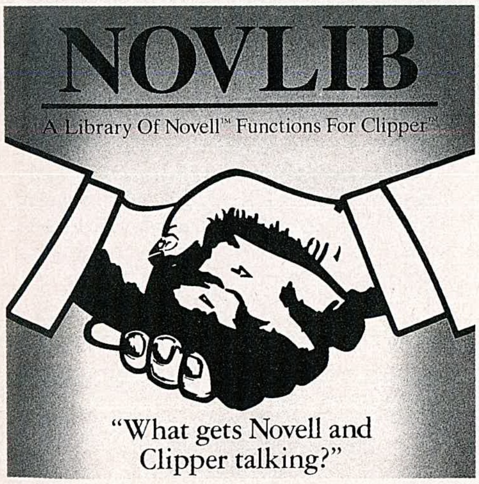
**ProVision WINDOW** Un moyen simple de créer des applications CLIPPER norme CUA. Initiez vous aux objets et à la programmation évènementielle. ProVision se charge de gérer l'interface utilisateur (menus, souris, windowing : fenêtre de taille modifiables, icônes.) **Frs 2 250 HT**

**BARCODE LIBRARY Frs 3 750 HT.**
**CLTEXT** Un vrai traitement de texte pour une application sous CLIPPER Frs 2 250 HT.
CLREPORT Un éditeur de rapport simple et puissant Frs 2 100 HT.
**BLINKER** V.F. Gain de mémoire, 15 fois plus rapide LE LINKER indispensable. Frs 2 850 HT.
**NOVLIB** V.F. Les 190 fonctions pour NOVELL Frs 2 890 HT.
**TELEPATHY** V.F. La télécommunication et CLIPPER Frs 2650 HT.
**AQUARIUM** Journal électronique sur CLIPPER 5.01 entre 60 et 80 pages mensuelles.
**CLIPWKS** de Lotus ou QUATTRO VERS CLIPPER ET VICE VERSA avec moins de 10 k Frs 1 750 HT.
Port et emballage 100 F sur nos produits.

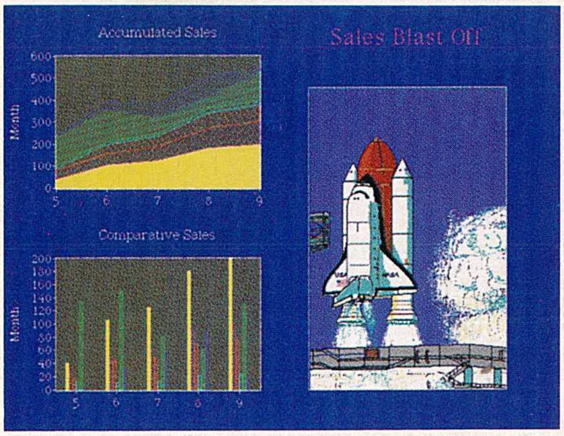
**SEMINAIRE DANS VOTRE REGION : 7 personnes maximum**

- CLIPPER 5.01 DEBUTANT ET EXPERT
- CLIPPER et la philosophie Windows
- CLIPPER ET LA TELECOMMUNICATION
- CLIPPER PAR L'OBJET
- RESEAU NOVELL
- C++ TURBO PASCAL

**FORMATION SUR SITE**
*Demandez notre catalogue*




**NOVLIB V.F. Frs 2 890 HT.**
Les 190 fonctions pour NOVELL

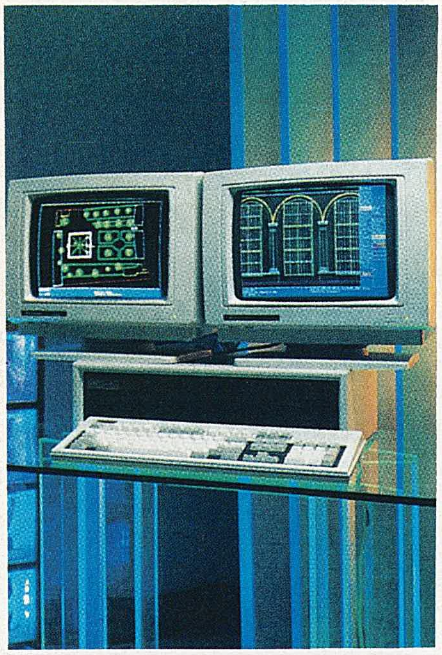



**FLIPPER Frs 2 650 HT**
La lib graphique la plus complète
pour votre application CLIPPER

**L.D.Q.S**

5 rue Eugene Emanuel 06000 NICE. FRANCE.
☎ : (33). 93.16.09.12 - fax : (33). 93.88.08.90.

# Notion d'algorithmique Les structures de données évoluées



**Dans cette série d'articles, je vous propose de découvrir certaines structures de données évoluées très couramment utilisées en informatique ; nous étudierons le fonctionnement et l'implémentation en langage C des piles, des files, des listes chaînées et des arbres binaires.**

Frédéric BERTON

J'ai volontairement choisi le langage C pour réaliser les différents développements que nous serons amenés à réaliser car ce langage est particulièrement bien adapté à la définition de ce type de données.

Toutefois, il sera très facile d'effectuer des conversions vers d'autres langages évolués comme Pascal ou ADA.

Mais maintenant passons aux choses sérieuses, la première structure de données que nous allons étudier est la pile.

## La pile

La notion de pile est très souvent utilisée dans la vie courante, de la pile d'assiettes à la pile de courriers en retard en passant par la pile de dossiers ; les exemples ne manquent pas. Le principe de fonctionnement d'une pile est toujours le même quelque soit sa nature, on empilera des dossiers les uns sur les autres, puis on les dépilera en commençant par le dossier qui se trouve sur le haut de la pile. Ce qui revient à dire que le dernier dossier empilé sera le premier dépilé.

La pile est une structure de données fondamentale en informatique, on l'utilisera pour stocker des paramètres que l'on passe à un sous-programme pour garder la trace des transactions effectuées dans une gestion de transaction, etc...

Dans les piles, l'ajout et la suppression de données se fait toujours par la même extrémité que l'on appelle le sommet (on verra plus loin que d'autres structures de données permettent d'effectuer des opérations équivalentes n'importe où dans la pile, qui ne s'appellera plus une pile mais une liste). Les piles gérées comme nous venons de le voir sont communément appelées pile LIFO pour Last In First Out ou plus simplement Dernier Entrée Premier Sortie (DEPS).

Les actions possibles sur une pile sont d'empiler un élément au sommet de celle-ci, dépiler un élément du sommet de la pile (1), tester si la pile est vide.

On peut bien sûr ajouter bien d'autres fonctionnalités comme dupliquer, comparer, trier, etc... Mais ceci sort du cadre de cet article, et puis il faudra bien que vous occupiez vos longues soirées d'hiver.

Pour implémenter les piles, on peut utiliser deux représentations distinctes. La représentation contiguë sous forme de tableau, ou la représentation chaînée mettant en œuvre les pointeurs pour faire un chaînage des éléments de la pile. Cette dernière méthode bien qu'un peu plus complexe à mettre en œuvre présente l'avantage d'être dynamique, et donc de ne pas être limitée par un nombre d'éléments figés comme dans un tableau.

## Implémentation de la représentation contiguë d'une pile

La représentation d'une pile en utilisant un tableau est certainement la méthode la plus simple que l'on puisse mettre en œuvre, étant donné la simplicité d'utilisation d'un tableau.

Malheureusement elle souffre des limites inhérentes aux tableaux. Effectivement on sera obligé de connaître exactement la taille de la pile lors de sa déclaration dans le programme ; or bien souvent le programmeur n'a qu'une idée très approximative de la taille qu'il faudra pour sa pile.

Il sera donc obligé de surdimensionner la taille de celle-ci pour ne pas risquer de dépasser la capacité de sa pile (provoquant inexorablement un "core dump de mort"). Il est évident que la simplification du programme se fait au détriment de ses performances surtout au niveau de la taille de l'exécutable.

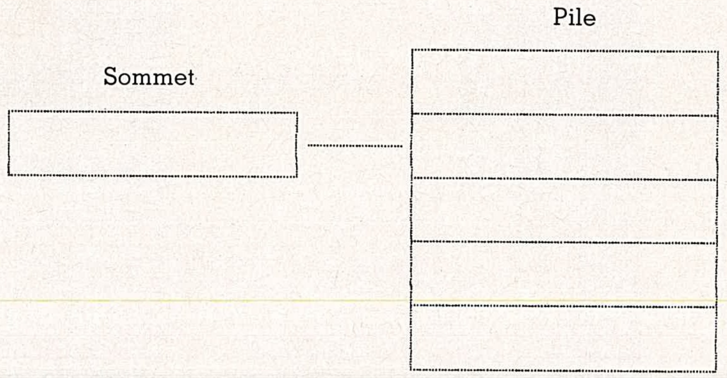



Figure 1 : Représentation contiguë d'une pile.

Comme le montre la figure 1, l'implémentation d'une pile par un tableau indicé, se fait simplement en déclarant un tableau d'une taille déterminée, que l'on appellera Pile. C'est dans ce tableau que seront stockées les données de notre pile. Il faudra parallèlement maintenir un index appelé Sommet qui donnera la position du sommet de la pile.

Lors de la définition de la fonction Empiler() on prendra soin d'ajouter un test destiné à éviter le débordement de la pile. La position d'insertion dans le tableau est obtenue en incrémentant de 1 la valeur de Sommet soit : Pos = Sommet + 1.

Pour dépiler un élément, il suffit d'adresser une case du tableau en utilisant comme index la valeur de Sommet soit : Elément = Pile[Sommet[. En n'oubliant pas de décrémenter Sommet de 1. Comme vous vous en doutez, la fonction Dépiler() devra contenir un test évitant de dépiler plus bas que la limite inférieure de la pile (sinon vous tombez).

Vous pourrez constater que dans une représentation contiguë les éléments dépilés ne sont perdus que logiquement, physiquement ils sont toujours stockés dans la pile au même endroit. Ils ne seront réellement perdus que si vous empilez de nouvelles informations, écrasant ainsi les anciennes.

A l'issue de ses quelques explications, il est évident que tout le travail de gestion de la pile est réalisé grâce à la bonne utilisation de la variable Sommet.

Les fichiers PILECONT.C et PILECONT.H vous présentent un exemple d'implémentation de la représentation contiguë d'une pile.

Le fichier PILECONT.H contient la déclaration du type de données pile contiguë (PileCont__t) qui est une structure appelée PileCont__s. Cette structure correspond à la figure 1 ; elle contient un entier non signé (Sommet) qui donne l'adresse du sommet de la pile (Pile) qui est représentée par un tableau indexé.

Le nombre d'éléments de Pile est spécifié par la constante MAXPILE. Attention l'élément 0 du tableau n'est pas utilisé. Lorsque la variable Sommet contient 0 la pile est vide. Ce fichier contient également les déclarations des différents codes erreurs que peuvent retourner les fonctions définies, et les déclarations des prototypes de fonctions.

## Implémentation de la représentation chaînée d'une pile

Voyons maintenant comment implémenter une pile grâce à la représentation chaînée. Beaucoup de détracteurs (2) du langage C le critiquent car ils prétendent que l'utilisation des pointeurs rend ce langage trop dur à lire et difficile à maintenir. J'espère que les lignes qui vont suivre vous persuaderont du contraire.

La figure 2 reprend le même schéma que la figure 1 sauf que les différents éléments de la pile sont liés les uns aux autres par des pointeurs (symbolisés par des cases dont partent des flèches).

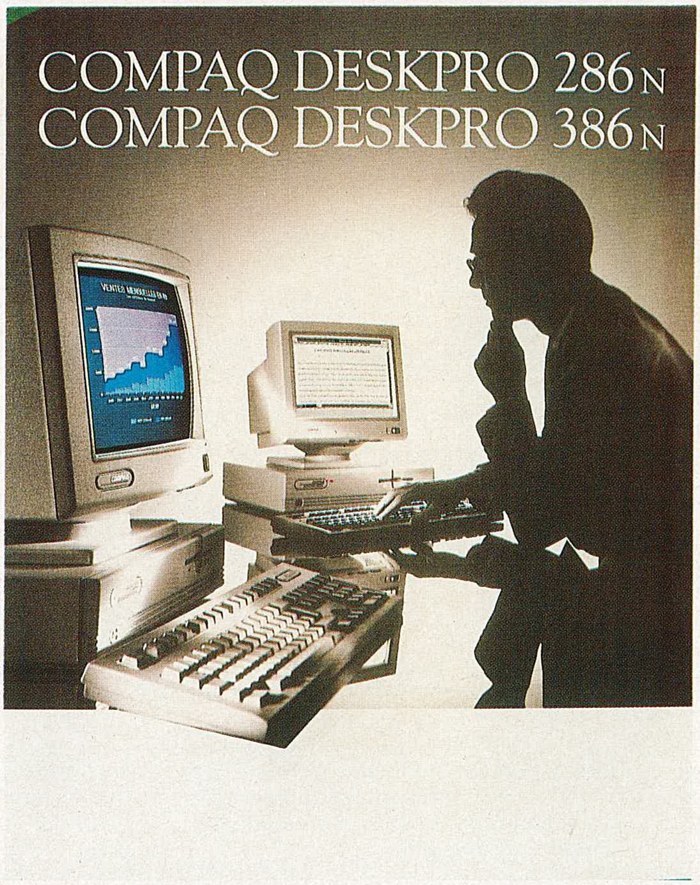
Le principe de fonctionnement est le même que celui que nous venons de voir pour la pile continguë, à quelques différences près.

La structure PileChaîne__t contient le nombre d'éléments total de la pile dans une variable de type entier non signé NbreElem (si cette valeur est 0 alors la pile est vide), et un pointeur de type Nœud__t * pSommet (si ce pointeur est égal à NULL alors la pile n'existe pas, ou est vide pour l'instant) sur l'élément de la pile qui est au sommet de celle-ci.

Les différents éléments de la pile quant à eux sont de type Nœud__t. Un nœud contient une donnée Donnée (dans notre exemple un entier long) et un pointeur sur le nœud suivant de type Nœud__t * pNext. Chaque élément de la pile pointe sur l'élément suivant assurant ainsi le chaînage des éléments entre eux.

Pour manipuler la pile on utilisera toujours une variable de type PileChaîne__t qui contient le pointeur pSommet sur le sommet de la pile. On n'utilise donc jamais directement les éléments de la pile, on passe toujours par le pointeur sur le sommet comme dans la Représentation contiguë.

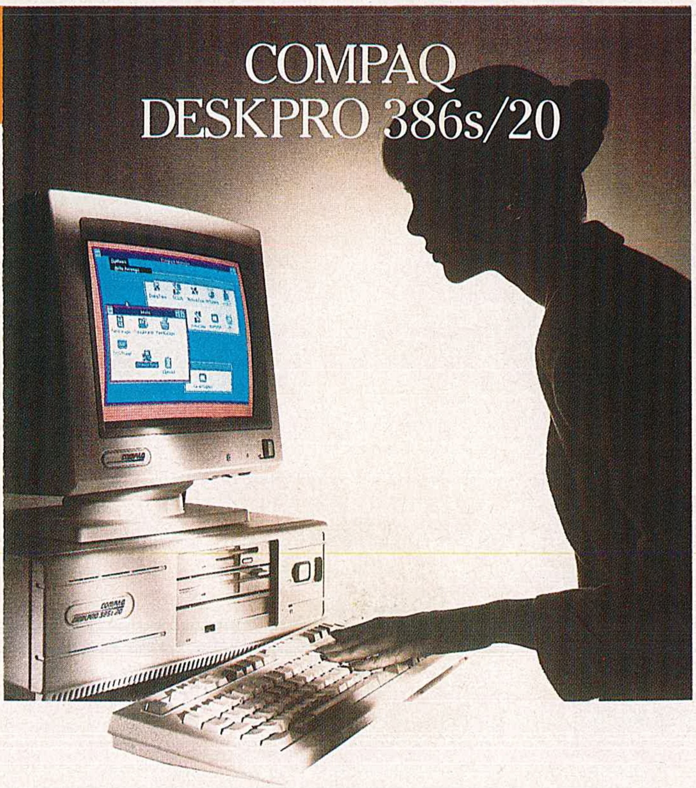
Equipés respectivement des microprocesseurs 286 à 12 MHz et 386SX à 16 MHz, le **COMPAQ DESKPRO 286N** et le **COMPAQ DESKPRO 386N** offrent en standard toutes les fonctionnalités des micro-ordinateurs professionnels : 3 connecteurs d'extension (2 connecteurs 8/16 bits ISA et 1 connecteur dédié à la mémoire vive rapide), 1 Mo de mémoire vive extensible à 13 Mo (286N) et 16 Mo (386N), jusqu'à 120 Mo de mémoire de masse en interne, 2 emplacements de mémoire de masse, un nouveau contrôleur graphique VGA accéléré avec support du mode texte 132 colonnes.




**COMPAQ DESKPRO 286 N**
**COMPAQ DESKPRO 386 N**

Basé sur une architecture évolutive en termes de processeur et de mémoire de masse, le **COMPAQ SYSTEMPRO** a été le premier micro-ordinateur à offrir un mode multi-processeur et un système multidisque. Il évolue ainsi avec les besoins des utilisateurs et préserve leurs investissements. Cette nouvelle génération d'ordinateurs a connu un grand succès : de nombreuses entreprises sont en train d'adopter le **COMPAQ SYSTEMPRO** pour faire évoluer leurs sites centraux et leur mini-informatique vers une informatique ouverte et au standard de l'industrie.

**COMPAQ SYSTEMPRO**

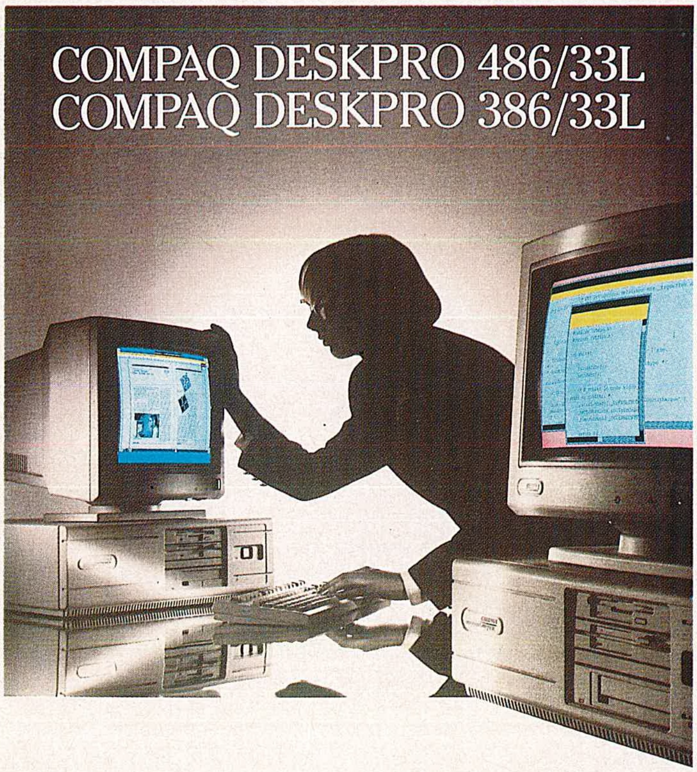



Equipé du dernier microprocesseur INTEL 80386SX à 20 MHz, le **COMPAQ DESKPRO 386s/20** est disponible en 3 modèles : 1,60 et 120 (disques durs avec temps d'accès inférieur à 19 ms). Outre sa mémoire cache ultra-rapide, le **COMPAQ DESKPRO 386s/20** est doté de nombreuses innovations. Son contrôleur VGA est ainsi accéléré pour offrir un maximum de performances en environnement graphique. De plus, il supporte le mode texte 132 colonnes utilisé par un nombre grandissant de logiciels de type tableur, traitement de texte ou d'émulation terminal.

**COMPAQ DESKPRO 386s/20**

Le **COMPAQ DESKPRO 486/33L** est aujourd'hui le micro-ordinateur de bureau le plus puissant du monde. En plus d'un coprocesseur 80387 et de 8 Ko de mémoire cache intégrés au processeur i486, sa carte processeur comprend un cache de deuxième niveau de 128 Ko de mémoire ultra-rapide. Ce nouveau micro-ordinateur est conçu pour des applications spécialisées techniques telles que la CAO, le développement logiciel, les SGBD et les applications scientifiques.

Le **COMPAQ DESKPRO 386/33L** dispose d'une carte processeur qui est équipée du microprocesseur 386 à 33 MHz et de 64 Ko de mémoire cache. Il constitue l'outil idéal pour les applications spécialisées telles que la CAO, les SGBD, la modélisation et l'analyse financière, ainsi que la PAO.
Un programme d'échange de la carte processeur 386 contre la carte i486 va permettre d'assurer aux utilisateurs la pérennité de leurs investissements, tout en répondant à leurs besoins d'évolutivité.




# LA DEDICACE INFORMATIQUE

**COMPAQ**
**REVENDEUR AGREE**

173, rue de Vaugirard 75015 PARIS
Tél. 45.66.94.69
Fax : 43.06.35.18 - Télex 202 993

**COMPAQ DESKPRO 486/33L**
**COMPAQ DESKPRO 386/33L**

Création Parenthèse - Paris - 43 55 76 20

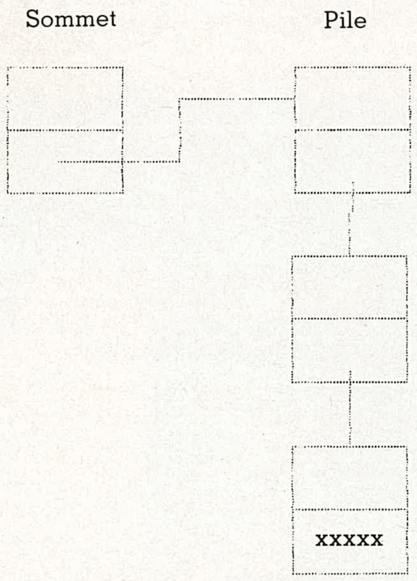



Figure 2 : Représentation chaînée d'une pile.

La fonction Empiler() définie dans PILE-CHAINE.C assure trois fonctions principales : premièrement elle alloue de la mémoire aux éléments à empiler en créant un pointeur de type Nœud_t *, deuxièmement elle assure le chaînage des éléments entre eux, et troisièmement elle effectue la gestion de l'en-tête de pile.

La fonction Dépiler() récupère la donnée qui se trouve au sommet de la pile en utilisant le pointeur sur le sommet (pSommet) spécifié dans l'en-tête. Le nombre d'éléments de la pile (NbreElem) est décrémenté, puis le pointeur sur le sommet est mis à jour pour pointer sur l'élément de la pile précédant le sommet. En dernier lieu, la mémoire allouée pour l'élément récupéré est libérée.

Contrairement à la représentation contiguë où les éléments dépilés ne sont perdus que logiquement. Dans la représentation chaînée, ces éléments sont perdus physiquement puisque la place mémoire qui leur est attribué par la fonction Empiler(), est restituée au système par la fonction Dépiler().

## En conclusion sur les piles

Nous venons de voir deux possibilités d'implémentation d'une pile. Comme vous l'avez sûrement remarqué la donnée que nous avons stocké dans notre pile était toujours un entier long, vous devez penser que pour une structure évoluée nous sommes bien limités. En fait, j'ai volontairement limité mes exemples à ce type de données pour simplifier mes démonstrations.

Dans une utilisation réelle la structure Nœud_s aurait plutôt la forme suivante.

```
struct Noeud_s
{
   void            *pData; /* Pointeur sur une donnée non typée */
   unsigned int    Size;   /* Taille de la donnée pointée par pData */
   struct Noeud_s *pNext; /* Pointeur sur le noeud suivant */
};
```

Il suffirait de faire quelques modifications dans les fonctions Empiler() et Dépiler() pour gérer ce type de structure. Nous aurons l'occasion d'en reparler lorsque nous étudierons les listes chaînées dans notre prochain article.

(1) Il est important de noter que l'action de dépiler une information entraîne la perte de celle-ci au niveau de la pile, sinon on ne peut pas dépiler puisque l'élément reste toujours sur la pile.

(2) Certainement des gens qui ont trop abusés du COBOL, qui comme tout le monde le sait, cause des dégâts irrémédiables, et, irréversibles au cerveau en cas cas d'utilisation prolongée. D'ailleurs ce langage n'est pas remboursé par la Sécurité Sociale.

```
PILECHAI.C                         Saturday, November 2, 1991 10:40 am        Page 1

/*****************************************************************************
 Nom fichier   : PILECHAINE.C
 Version ..... : 1.00
 Objet ....... : Implementation d'une pile en representation chaine
                 (par pointeur)
 Date cration : 18/11/90
 Date MAJ .... : 18/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>

#include "pilechaine.h"


/*==========================================================================*/
/*                              MYMALLOC()                                  */
/*==========================================================================*/
static char *MyMalloc( Taille )
unsigned Taille;
{
   char *Ptr;

   Ptr = (char *) malloc( Taille );
   if (Ptr == NULL)
      fprintf(stderr,"Plus de memoire dans MyMalloc()\n");

return Ptr;
}


/*==========================================================================*/
/*                              EMPILER()                                   */
/*==========================================================================*/
int  Empiler( pPile, Donnee )
PileChaine_t *pPile; /* Pointeur sur la structure pile */
long          Donnee; /* Donnee a placer dans la pile */
{
   Noeud_t *pPileElement;

   /*** Creation de l'element a ajouter dans la pile ***/
   pPileElement = (Noeud_t *) MyMalloc( sizeof(Noeud_t) );
   if ( pPileElement == NULL )
     return ER_OUTOFMEM;

   /*** MAJ de l'element est chainnage avec les autres elements ***/
   pPileElement->Donnee = Donnee;
   pPileElement->pNext  = pPile->pSommet;

   /*** MAJ de l'entete de pile ***/
   pPile->pSommet    = pPileElement;
   pPile->NbreElem++;

return ER_OK;
} /* fin de fct : Empiler() */


/*==========================================================================*/
/*                              DEPILER()                                   */


PILECHAI.C                         Saturday, November 2, 1991 10:40 am        Page 2

/*==========================================================================*/
int  Depiler( pPile, pDonnee )
PileChaine_t *pPile;   /* Pointeur sur la structure pile */
long         *pDonnee; /* Donnee a sortir de la pile */
{
  int Rts = ER_OK;
  Noeud_t *pPileElement;

  if ( PileVide(pPile) )
  {
    /* La pile est vide erreur */
    *pDonnee = 0L;

    Rts = ER_PILEVIDE;
  } /* endif */
  else
  {
    /*** Recupere le sommet de la pile ***/
    pPileElement = pPile->pSommet;
    *pDonnee     = pPileElement->Donnee;

    /*** MAJ de l'entete de la pile ***/
    pPile->NbreElem--;
    pPile->pSommet = pPileElement->pNext;

    /*** liberation de la mmoire de l'element retirer  de la pile ***/
    free( pPileElement );

    Rts = ER_OK;
  } /* endelse */

return Rts;
} /* fin de fct : Depiler() */


/*==========================================================================*/
/*                              PILEVIDE()                                  */
/*==========================================================================*/
BOOL PileVide( pPile )
PileChaine_t *pPile;   /* Pointeur sur la structure pile */
{
  BOOL Rts = FAUX;

  if ( pPile->NbreElem == 0 ) Rts = VRAI;

return Rts;
} /* fin de fct : PileVide() */


/*****************************************************************************
 Fin du fichier   : PILECHAINE.C
*****************************************************************************/
```

```
PILECHAI.H                          Saturday, November 2, 1991 10:39 am                          Page 1

#ifndef __PILECHAINE_H__
#define __PILECHAINE_H__ 1
/*****************************************************************************
 Nom fichier   : PILECHAINE.H
 Version ..... : 1.00
 Objet ....... : Definition des types de donnes de l'implementation d'une
                 pile en representation chaine (par pointeur)
 Date cration : 18/11/90
 Date MAJ .... : 18/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
*****************************************************************************/


/*---------------------------------------------------------------------------*/
/*                          DEFINITION DES MACROS                            */
/*---------------------------------------------------------------------------*/
#define VRAI        1
#define FAUX        0
#define ER_OK       1
#define ER_PILEVIDE -1
#define ER_OUTOFMEM -2


/*---------------------------------------------------------------------------*/
/*                     DECLARATION DES TYPES DE DONNEES                      */
/*---------------------------------------------------------------------------*/
typedef int BOOL;

struct Noeud_s
{
   long           Donnee; /* Donnee a stocker dans la pile */
   struct Noeud_s *pNext; /* Pointeur sur l'element suivant dans la pile */
};
typedef struct Noeud_s Noeud_t;

struct PileChaine_s
{
   unsigned int NbreElem; /* Nbombre d'element dans la pile */
   Noeud_t      *pSommet; /* Pointeur sur le sommet de la pile */
};
typedef struct PileChaine_s PileChaine_t; /* Type pile contigue */


/*---------------------------------------------------------------------------*/
/*                       DECLARATION DES PROTOTYPES                          */
/*---------------------------------------------------------------------------*/
int  Empiler( PileChaine_t *pPile, long Donnee );
int  Depiler( PileChaine_t *pPile, long *pDonnee );
BOOL PileVide( PileChaine_t *pPile );


/*****************************************************************************
 Fin du fichier   : PILECHAINE.H
*****************************************************************************/
#endif
```

```
PILECH_T.C                          Saturday, November 2, 1991 10:40 am                          Page 1

/*****************************************************************************
 Nom fichier   : PILECH_T.C
 Version ..... : 1.00
 Objet ....... : Test des fonctions de PILECHAINE.C
 Date cration : 18/11/90
 Date MAJ .... : 18/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
*****************************************************************************/

#include <stdio.h>
#include "pilechaine.h"


/*===========================================================================*/
/*                                 MAIN()                                    */
/*===========================================================================*/
/*
 * Description:
 *---------------------------------------------------------------------------*/
main()
{
   PileChaine_t MaPile;
   long Data;

   memset( &MaPile, 0, sizeof(PileChaine_t) );

   Empiler( &MaPile, 1 );

   Depiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Depiler( &MaPile, &Data );  printf("Data %d\n",Data);

   Empiler( &MaPile, 1 );
   Empiler( &MaPile, 2 );
   Empiler( &MaPile, 3 );
   Empiler( &MaPile, 4 );
   Empiler( &MaPile, 5 );
   Empiler( &MaPile, 6 );

   Depiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Depiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Depiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Depiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Depiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Depiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Depiler( &MaPile, &Data );  printf("Data %d\n",Data);

} /* fin de fct : main() */


/*****************************************************************************
 Fin du fichier   : PILECH_T.C
*****************************************************************************/
```

```
PILECONT.C                    Saturday, November 2, 1991 10:40 am                 Page 1


/*****************************************************************************
 Nom fichier   : PILECONT.C
 Version ..... : 1.00
 Objet ....... : Implementation d'une pile en representation contigue
                 (par tableau indice)
 Date cration : 15/11/90
 Date MAJ .... : 15/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
*****************************************************************************/

#include "pilecont.h"


/*==========================================================================*/
/*                              EMPILER()                                   */
/*==========================================================================*/
int  Empiler( pPile, Donnee )
PileCont_t *pPile; /* Pointeur sur la structure pile */
long       Donnee; /* Donnee a placer dans la pile */
{
   int Rts = ER_OK;
   int Pos = pPile->Sommet;

   Pos++;

   if ( Pos < MAXPILE )
   {
      /* Cas normal on insert une donnee dans la pile */
      pPile->Pile[Pos] = Donnee;
      pPile->Sommet    = Pos;

      Rts = ER_OK;
   } /* endif */
   else
   {
      /* On essaye d'ecrire en deshors de la pile donc erreur */
      Rts = ER_OUTOFPILE;
   } /* endelse */

return Rts;
} /* fin de fct : Empiler() */


/*==========================================================================*/
/*                              DEPILER()                                   */
/*==========================================================================*/
int  Depiler( pPile, pDonnee )
PileCont_t *pPile;
long       *pDonnee;
{
   int Rts = ER_OK;

   if ( PileVide(pPile) )
   {
      /* La pile est vide erreur */
      *pDonnee = 0;

      Rts = ER_PILEVIDE;
   } /* endif */
   else
```

```
PILECONT.C                    Saturday, November 2, 1991 10:40 am                 Page 2


   {
      /* Cas normal on insert une donnee dans la pile */
      *pDonnee = pPile->Pile[pPile->Sommet];
      pPile->Sommet--;

      Rts = ER_OK;
   } /* endelse */

return Rts;
} /* fin de fct : Depiler() */


/*==========================================================================*/
/*                              PILEVIDE()                                  */
/*==========================================================================*/
BOOL PileVide( pPile )
PileCont_t *pPile;
{
   BOOL Rts = FAUX;

   if ( pPile->Sommet == 0 ) Rts = VRAI;

return Rts;
} /* fin de fct : PileVide() */


/*****************************************************************************
 Fin du fichier   : PILECONT.C
*****************************************************************************/
```

```
PILECONT.H                        Saturday, November 2, 1991 10:39 am          Page 1


#ifndef __PILECONT_H__
#define __PILECONT_H__ 1
/******************************************************************************
 Nom fichier   : PILECONT.H
 Version ..... : 1.00
 Objet ....... : Definition des types de donnes de l'implementation d'une
                 pile en representation contigue (par tableau indice)
 Date cration : 15/11/90
 Date MAJ .... : 15/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
******************************************************************************/

/*---------------------------------------------------------------------------*/
/*                          DEFINITION DES DEFINES                           */
/*---------------------------------------------------------------------------*/
#define MAXPILE 5           /* Taille maximum de la pile */

/*---------------------------------------------------------------------------*/
/*                          DEFINITION DES MACROS                            */
/*---------------------------------------------------------------------------*/
#define VRAI        1
#define FAUX        0
#define ER_OK       1
#define ER_PILEVIDE  -1
#define ER_OUTOFPILE -2

/*---------------------------------------------------------------------------*/
/*                     DECLARATION DES TYPES DE DONNEES                      */
/*---------------------------------------------------------------------------*/
typedef int BOOL;
struct PileCont_s
{
   unsigned int Sommet;            /* Sommet de la pile */
   long         Pile[MAXPILE];     /* La pile elle meme */
};

typedef struct PileCont_s PileCont_t; /* Type pile contigue */

/*---------------------------------------------------------------------------*/
/*                        DECLARATION DES PROTOTYPES                         */
/*---------------------------------------------------------------------------*/
int  Empiler( PileCont_t *pPile, long Donnee );
int  Depiler( PileCont_t *pPile, long *pDonnee );
BOOL PileVide( PileCont_t *pPile );


/******************************************************************************
 Fin du fichier   : PILECONT.H
******************************************************************************/
#endif
```

```
PILEC_T.C                         Saturday, November 2, 1991 10:40 am          Page 1

/******************************************************************************
 Nom fichier   : PILEC_T.C
 Version ..... : 1.00
 Objet ....... : Test des fonctions de PILECONT.C
 Date cration : 15/11/90
 Date MAJ .... : 15/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
******************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "pilecont.h"

void TRACEPILE( pPile )
PileCont_t *pPile;
{
   char Rep;
   int i;

   printf("Trace pile (o/n) : "); Rep = getche(); putchar('\n');
   if ( Rep == 'o' )
   {
      printf("Sommet  = %d\n",pPile->Sommet);
      for (i=1; i<MAXPILE; i++)
      {
         printf("Pile[%d] = %d\n",i,pPile->Pile[i]);
      } /* endfor */
   } /* endif */
}
/*===========================================================================*/
/*                                 MAIN()                                    */
/*===========================================================================*/
/*
 * Description:
 *---------------------------------------------------------------------------*/
main()
{
   PileCont_t MaPile;
   long Data;

   memset( &MaPile, 0, sizeof(PileCont_t) );

   Empiler( &MaPile, 1 );     TRACEPILE( &MaPile );
   Depiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data %d\n",Data);
   Depiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data %d\n",Data);
   Empiler( &MaPile, 2 );     TRACEPILE( &MaPile );
   Empiler( &MaPile, 3 );     TRACEPILE( &MaPile );
   Empiler( &MaPile, 4 );     TRACEPILE( &MaPile );
   Empiler( &MaPile, 5 );     TRACEPILE( &MaPile );
   Empiler( &MaPile, 6 );     TRACEPILE( &MaPile );
   Depiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data %d\n",Data);
   Depiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data %d\n",Data);
   Depiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data %d\n",Data);
   Depiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data %d\n",Data);
   Depiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data %d\n",Data);

} /* fin de fct : main() */

/******************************************************************************
 Fin du fichier   : PILEC_T.C
******************************************************************************/
```

## Arrêtez de développer ce qui existe déjà...

# COMPILEZ... C'EST GAGNÉ !

## FINI

La programmation sans fin
Les séances de debugging
Les essais interminables

### AGENDA Multi-Utilisateur

Cette gestion des rendez-vous très visuelle, vous sera très vite indispensable. Elle gère l'utilisation en multiposte, permet la programmation des rendez-vous, de rechercher des plages de temps libres. Possibilité de couper-coller des rendez-vous.

- Version exécutable bridée ......... 150 F H.T.
- Version exécutable ............... 649 F H.T.
- Source en TP 6.0 ................ 2490 F H.T.

### CODE A BARRE

Impression des codes à barres au standard EAN 13, 2/5 entrelacé et Alpha 39.

- Version exécutable ............... 649 F H.T.
- Source en TP 5.5 ................ 1990 F H.T.

### GESTION DES STOCKS FACTURATION - CAISSE

Gestion des fichiers produits, clients, fournisseurs. Permet d'éditer vos devis, factures, avoirs, bons de commande, bons de livraison. Gère les ventes au "comptoir", édition des tickets de caisse, du ticket récapitulatif. La version Stock 4 comporte en plus une gestion du multiposte, une aide contextuelle, une gestion des représentants, les taxes parafiscales, un générateur d'état, des éditions paramétrables, des sélections multicritères pour toutes les éditions (listes, courriers, fiches récapitulatives, etc...), import-export de fichiers en ASCII.

- Version exécutable de Stock 4 bridée 150 F H.T.
- Version exécutable de Stock 3 ..... 430 F H.T.
- Version exécutable de Stock 4 ..... 1850 F H.T.
- Source Stock 3 en TP 3.0 ......... 2990 F H.T.
- Source Stock 4 en TP 5.5 ......... 5990 F H.T.

### GESTION FICHIER COMMERCIAL

Gestion d'un fichier de prospection. Grâce à son générateur d'état intégré, il vous offre la possibilité de mailing ciblé et personnalisé et d'édition d'étiquettes. Le programme gère le multiposte, comporte une aide contextuelle, etc...

- Version exécutable bridée ......... 150 F H.T.
- Version exécutable ............... 1260 F H.T.
- Source en TP 5.5 ................ 3990 F H.T.

**SIMASOFT** le concepteur et l'éditeur de logiciels le plus souple du marché, vous permet désormais d'acquérir les sources de plusieurs de ses programmes (écrits en Turbo Pascal). Vous aurez alors la permission de revendre vos nouvelles versions compilées sans limitations quantitatives et sans payer de redevances. Pour ceux qui ne possèdent pas le Turbo Pascal nous pouvons aussi vous personnaliser ces logiciels.

Nos logiciels sont utilisés par milliers dans de nombreux domaines (PME-PMI, artisans, commerçants, professions libérales, médicales, para-médicales), d'où une garantie de fonctionnement irréprochable.

Si vous ne connaissez pas nos logiciels, des versions bridées (déductible d'un achat ultérieur) vous permettront de juger de leurs qualités.

Nos logiciels fonctionnent sous MS-DOS (Version > = 3.0) sur un ordinateur compatible PC avec 640 K de RAM et un disque dur. Pour compiler les sources, vous devez posséder le compilateur Turbo Pascal de Borland.

Nos prix sont HT (TVA 18.6%). Une facture justificative est jointe aux envois. Les commandes doivent impérativement être accompagnées du règlement du montant total (logiciel + frais de port) soit par chèque, soit par mandat. Les frais de port sont de 29 F par commande, ou de 50 F par envoi en contre-remboursement, ou de 100 F pour les DOM-TOM et l'exportation.

Rédigez votre bon de commande de manière lisible, en indiquant vos nom, prénoms, adresse complète, Numéro de téléphone, ainsi que le format des disquettes que vous utilisez. Datez, signez et envoyez votre commande à l'adresse ci-dessous :

SIMA SOFTWARE

12, RUE DE LA RÉPUBLIQUE
13420 GÉMENOS - TÉL. 42.32.09.32
FAX. : 42.72.66.54 ET 42.32.17.27

### COMPTABILITE GENERALE

Comptabilité générale multi-sociétés. Saisie des écritures par brouillard. Lettrage automatique et manuel. Editions des brouillards, journaux, grands-livres, balances, bilan, compte de résultat. La version II gère en plus le multiposte, des guides de saisies automatiques.

- Version exécutable bridée PROFIMAX II 150 F H.T.
- Version exécutable PROFIMAX I .... 449 F H.T.
- Version exécutable PROFIMAX II ... 1690 F H.T.
- Source PROFIMAX I en TP 3.0 ..... 3490 F H.T.
- Source PROFIMAX II en TP 5.5 .... 5990 F H.T.

### COMPTABILITE LIBERALE 2035

Comptabilité simplifiée pour les professions libérales (soumises ou non à la TVA). Saisie en mode recettes - dépenses.

- Version exécutable bridée ......... 150 F H.T.
- Version exécutable ............... 1690 F H.T.
- Source en TP 6.0 ................ 5990 F H.T.

### GESTION DE LA PAYE ET DU PERSONNEL

Paye multi-sociétés entièrement paramétrable. Calcul de la paye mensuelle, horaire, paye à l'envers. Editions des bulletins aux nouvelles normes. Gestion des congés payés, des repos compensateurs, calcul de la CSG. Editions du livre de paye, des fiches individuelles, des fiches récapitulatives, de l'état préparatoire à la DADS, des avis de virements. Gestion des règlements, des acomptes.

- Version exécutable bridée ......... 150 F H.T.
- Version exécutable ............... 1790 F H.T.
- Source en TP 5.5 ................ 5990 F H.T.

### SENATOR II GESTION COMPLETE D'ENTREPRISE

Reprend toutes les fonctionnalités du programme Stock 3 (GESTION DES STOCKS - FACTURATION - CAISSE), du programme Profimax I (COMPTABILITE GENERALE). De plus il contient un traitement de texte et une gestion de fichier commercial. Le tout est lié dans un ensemble cohérent et interactif (Transfert des factures, avoirs en comptabilité, transfert des prospect en gestion de stock, accès au traitement de texte pour toutes les éditions, Etc...

- Version exécutable ............... 1260 F H.T.
- Source en TP 3.0 ................ 6990 F H.T.

# VOUS ALLEZ GAGNER DE L'ARGENT

PAS DE REDEVANCES

# La Programmation Orientée Objet en PASCAL

**Depuis quelques années, la Programmation Orientée Objet gagne du terrain. Cette nouvelle méthode peut sembler quelque peu compliquée. Il n'en est rien. Comme toute méthode, il suffit de prendre le pli. Nous allons essayer de vous en expliquer toutes les ficelles. Le langage utilisé est le PASCAL (TURBO PASCAL Version 6).**

La Programmation Orientée Objet (POO) repose sur trois principes :

— l'encapsulation,
— l'héritage,
— le polymorphisme.

Nous aborderons peu à peu ces trois concepts. Pour l'instant, nous allons décortiquer le premier : l'encapsulation.

## L'encapsulation

Voici les mots-clés que nous allons aborder : objet, classe, méthode, constructor, destructor, new, dispose.

Dans la programmation PASCAL classique, le programmeur utilise souvent des enregistrements (RECORDS) pour représenter des données. Mais les procédures manipulant ces records n'ont aucun lien direct avec les données.

La POO consiste en une méthode de "représentation des éléments qui nous entoure". Elle récupère leurs caractéristiques (sous forme de données) et leurs comportements (sous forme de procédures).

Les procédures liées aux données prennent le nom de METHODES. Le rapprochement des données et des méthodes est appelé ENCAPSULATION.

Un OBJET (ou une CLASSE) est défini en PASCAL de la manière suivante : figure 1.

```
figure 1 : déclaration d'un objet.

type
    personne = object
                   nom    : string[20];
                   prenom : string[12];

                   constructor init(xnom, xprenom : string);
                   procedure affiche;
                   destructor done;
                end;
```

L'objet se décompose donc en deux parties : une partie déclaration des données (ici, nom et prénom) et une partie déclaration des méthodes devant manipuler ces données.

Vous remarquerez deux mots nouveaux : CONSTRUCTOR et DESTRUCTOR. Ces deux mots sont des noms de procédures particulières : elles servent au compilateur pour maintenir une ''table des méthodes virtuelles''. Retenez pour l'instant que le compilateur a besoin d'initialiser l'objet dans une table. Vous devez donc toujours utiliser en premier la procédure CONSTRUCTOR et en dernier la procédure DESTRUCTOR.

Les noms ''init'' et ''done'' donnés respectivement au constructeur et au destructeur ne sont pas obligatoires. Vous auriez très bien pu les appeler ''initialise'' et ''supprime''. Mais pour une raison de confort de lecture, que ce soit vos programmes ou ceux d'autres sources, nous nous obligerons à toujours les appeler ''init'' et ''done''.

Jusqu'ici, nous ne vous avons pas montré quelque chose de spectaculaire. Vous pouvez penser la Programmation Orientée Objet ne fait que déclarer des procédures d'une manière différente de la manière classique. Il n'en est rien. Ce que vous avez découvert jusqu'ici n'est que la partie émergée de l'iceberg.

Pour le moment, nous n'avons montré que la déclaration. Mais comment écrire les procédures, et comment utiliser des variables objet ? Regardez la figure 2 : elle vous montre comment écrire les méthodes.

figure 2 : écriture des méthodes de l'objet personne.

```
constructor personne.init(xnom, xprenom : string);
begin
   nom    := xnom;
   prenom := xprenom;
end;

procedure personne.affiche;
begin
   writeln('nom     : ',nom);
   writeln('prénom : ',prenom);
end;

destructor personne.done;
begin
end;
```

Rappelez-vous qu'un objet doit toujours être INITIALISE, et toujours REFERME par les méthodes constructor et destructor. Même si vous pensez que la méthode destructor ''personne.done'' ne fait rien ici, il n'en est rien : cela permet de libérer l'objet dans la table des méthodes virtuelles.

Vous voyez que pour distinguer la procédure appartenant à un objet, il suffit de préciser le nom de l'objet suivi d'un point, juste avant le nom de la procédure (ou méthode). Ainsi, il ne pourra y avoir de confusion entre un constructeur ''personne.init'' et un constructeur ''client.init''. L'objet personne dispose de sa propre méthode d'initialisation.

La déclaration de variables et l'utilisation de celles-ci seront écrites de la façon suivante : figure 3.

figure 3 : déclaration et utilisation de variables objet.

```
 var
   pers1 : personne;
   pers2 : personne;

 begin
   pers1.init('Montbazet','François');
   pers2.init('Martin','Pierre');

   pers1.affiche;
   pers2.affiche;

   pers1.done;
   pers2.done; end;
```

En programmation Pascal classique, on peut déclarer un pointeur sur un enregistrement. Avec la Programmation Orientée Objet, on peut pratiquement de la même manière déclarer un pointeur sur un objet. Voici comment : figure 4.

```
figure 4 : utilisation des pointeurs avec les objets.
 type
   ptr_personne = ^personne;

 var
   pers1 : ^personne;
   pers2 : ptr_personne;

 begin
   new(pers1);
   pers1^.init('Montbazet','François');
   pers2 := new(ptr_personne,init('Martin','Pierre'));

   pers1^.affiche;
   pers2^.affiche;

   pers1^.done;
   dispose(pers1);
   dispose(pers2,done);
 end.
```

Vous voyez que le Pascal accepte une nouvelle syntaxe pour les procédures new et dispose. On peut même servir de new en tant que fonction renvoyant une adresse pour un pointeur.

Vous pouvez avec cette nouvelle syntaxe faire d'une pierre deux coups : aller de la place mémoire pour le pointeur sur l'objet, et initialiser l'objet en question. Pour dispose et done vous utilisez le même principe.

Nous avons fait une approche rapide de l'encapsulation. Voyons maintenant l'héritage.

## L'héritage

Commençons par un exemple.

Maintenant que nous avons déclaré un objet personne, nous avons besoin de déclarer un objet client. Ce qui caractérise un client, c'est par exemple, son numéro, son nom, son prénom et le montant global de ses dettes. Vous remarquerez que client est une personne disposant d'un numéro et d'un montant.

Vous voyez où nous voulons en venir : client va être un nouvel objet qui va HERITER de personne. La déclaration de client se fera de la sorte : figure 5.

```
figure 5 : déclaration de client, héritant de personne.

type
    client = object(personne)
                  numero   : integer;
                  montant : real;

                  constructor init(xnom, xprenom : string;
                                   xnumero       : integer;
                                   xmontant      : real);
                  procedure affiche;
               end;
```

Les procédures rattachées à client sont : figure 6.

```
figure 6 : écriture des méthodes de l'objet client.

constructor client.init(xnom, xprenom : string;
                        xnumero       : integer;
                        xmontant      : real);
begin
   personne.init(xnom,xprenom);
   numero  := xnumero;
   montant := xmontant;
end;

procedure client.affiche;
begin
   writeln('numero : ',numero);
   personne.affiche;
   writeln('dettes : ',montant);
end;
```

Les variables nom et prénom, appartenant à l'objet personne, n'ont pas été redéfinies. Pourtant, client dispose bien de celles-ci...

Par contre, les trois méthodes ne sont pas redéfinies : done n'apparaît pas dans cet exemple. En effet, "client" se servira du destructor de l'objet "personne", parce que "client" n'a ici, pas besoin d'un destructeur spécial.

La programmation object va donc vous obliger, et ce n'est pas un mal, à réfléchir à la structure d'un objet avant de le déclarer. Comme tout programme doit être analysé avant d'écrire le source, tout objet doit être bien pensé. Il faut bien déterminer ce qui caractérise un objet (les variables) et comment il fonctionne (les méthodes).

Nous voyons par cet exemple que l'héritage permet à un objet de bénéficier des connaissances de ses ancêtres : "personne" sait comment s'initialiser, s'afficher. "Client" sait donc le faire.

Voici maintenant un exemple de ce qui est déconseillé de faire (que ce soit en Programmation Orientée Objet ou en programmation classique).

Lorqu'un objet est "figé", c'est-à-dire que vous estimez que sa structure ne changera pas, ou peu, vous pouvez commencer à le "décliner" en série de sous-ensembles : nous avons déclaré "client", nous allons maintenant déclarer "employé". De même que "client" était une personne ayant ses caractéristiques propres, "employé" sera une personne (héritage) qui disposera d'un matricule et d'un numéro de Sécurité Sociale. Nous allons déclarer : figure 7.

Vous voulez modifier l'objet "Personne", en lui rajoutant une variable "adresse". Personne aura donc trois variables : nom, prénom, adresse. Vous devrez donc modifier la procédure d'initialisation. Si personne était un objet qui n'avait aucun descendant, cela ne serait pas trop ennuyeux. Par contre, dès que l'on touche à un ancêtre, il y a fort à parier que l'on doit retoucher à tous les descendants. Cela peut aller très loin...

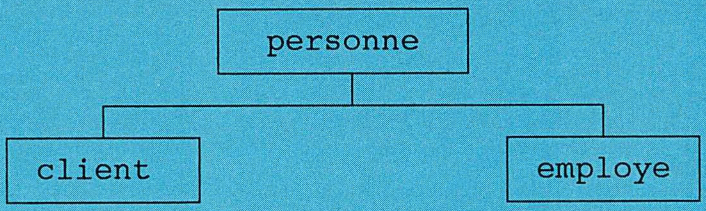
```
figure 7 : déclaration d'employé descendant de personne

 type  employe = object(personne)
                   matricule : integer;
                   secu_soc  : string[11];

                   constructor init(xnom,xprenom : string;
                                    xmatricule   : integer;
                                    xsecu_soc    : string);
                   procedure affiche;
                end;
```

L'écriture des procédures rattachées à "employé" sera : figure 8.

```
figure 8 : écriture des méthodes de l'objet employé.

constructor employe.init(xnom, xprenom : string;
                         xmatricule    : integer;
                         xsecu_soc     : string);
begin
   personne.init(xnom,xprenom);
   matricule := xmatricule;
   secu_soc  := xsecu_soc;
end;

procedure employe.affiche;
begin
   writeln('matricule : ',matricule);
   personne.affiche;
   writeln('code secu : ',secu_soc);
end;
```

Nous aurons donc le schéma suivant : figure 9.

figure 9 : l'arborescence de l'objet personne et de ses descendants




Attention : Pour que cela marche, nous devrons préciser à l'objet "personne" que certaines procédures ont été redéfinies. Dans l'exemple de "client" et "employé", la procédure "affiche" a été redéfinie. Aussi, nous allons rajouter un nouveau mot-clé : VIRTUAL, dans la déclaration de "personne". Voici donc comment écrire un objet dont on veut redéfinir des méthodes chez ses descendants : figure 10.

```
figure 10  : déclaration d'un objet ayant des procédures
             redéfinissables.

type
    personne = object
                  nom    : string[20];
                  prenom : string[12];

                  constructor init(xnom, xprenom : string);
                  procedure affiche;  virtual;
                  destructor done;    virtual;
               end;
```

# ARTOOL UN LOGICIEL DE D.A.O. POUR PC ET CARTES EGA OU VGA

***ARTOOL possède toutes les fonctions pour exploiter pleinement les possibilités graphiques de votre PC muni d'une carte EGA ou VGA. Il fonctionne en mode BITMAP, et gère toute la palette de couleurs.***

Spécifications techniques :

☐ Fonctionnement en 16 couleurs, 640×350 pour EGA et 640×480 pour VGA.

☐ Dessin à ''main levée'', tracé de droites, de cercles et de boîtes.

☐ Ajustement d'une courbe par la méthode de Bézier .

☐ Fonctions avancées : loupe, miroir, coloriage, texte.

☐ Gestion de la palette, des fichiers images et de l'imprimante.

ARTOOL fonctionne sur tous compatibles PC muni d'une souris et d'une carte graphique EGA ou VGA. Un minimum de 640 Ko ainsi qu'un disque dur sont vivement recommandés. Des impressions graphiques peuvent être obtenues sur des imprimantes compatibles IBM-GRAPH ou EPSON.

*Fourni avec les sources en langage PASCAL, vous pourrez adapter ARTOOL à vos besoins et à votre configuration.*

ARTOOL est disponible sur disquette 5'' 1/4 (360 Ko), au prix de 450 F (TTC)

---

BON DE COMMANDE
à adresser à La Revue de l'Utilisateur PC
2 rue d'Amsterdam
75009 Paris - Tél. : 42.81.54.27

Nom ..........................................................

Adresse ..........................................................

Code Postal .............................. Ville .............................. Téléphone ..............................

Veuillez m'envoyer ..... logiciel(s) ARTOOL, au prix unitaire de 450 F TTC.

Ci-joint un chèque de ......... F.

En règlement à l'ordre de Softin Communications.

## COUIC

**L'outil à désassembler**

Que vous **chassiez le virus** ou que vous cherchiez à comprendre comment fonctionne un programme **COM** ou **EXE**, que vous soyez programmeur professionnel ou débutant, il vous faut un bon **désassembleur immédiat**, véritable **fenêtre ouverte** sur le code machine.

— Vous devez pouvoir **désassembler un programme** en ROM, en RAM sur disque ou sur disquette en spécifiant une adresse, **un numéro d'interruption**, un nom de fichier ou un **secteur** et faire défiler le listing de désassemblage ou de dumping en avant et en arrière.

— Vous devez disposer d'instructions **puissantes** de localisation automatique d'**interruption**, de **suivi de jump**, ou de **recherches** de chaînes et de données.

— Vous devez disposer de commandes suffisamment simples pour être apprises et manipulées sans difficulté mais assez efficaces pour faciliter la lecture du code et effectuer des **marquages** dans les zones de données.

— Vous devez pouvoir générer des **fichiers séquentiels** utilisés par les désassembleurs symboliques.

— Vous devez disposer du **code source commenté** de votre désassembleur de façon à l'adapter à vos besoins spécifiques.

**COUIC-DESAS permet tout cela** et plus encore grâce à ses fonctions spéciales et ses tables descriptives d'interruptions.

**Exceptionnellement COUIC est vendu avec ses sources en langage ''C''** : plus de **9 000 lignes** de code réutilisable dans vos applications et sa documentation pour seulement **340 F (TTC)**.

**COUPON**
**à adresser à La Revue de l'Utilisateur de l'IBM PC**
**2, rue d'Amsterdam**
**75009 Paris**
**Tél. : 47.64.07.57. - 42.67.93.80.**

**Nom** ..........................................................

**Adresse**

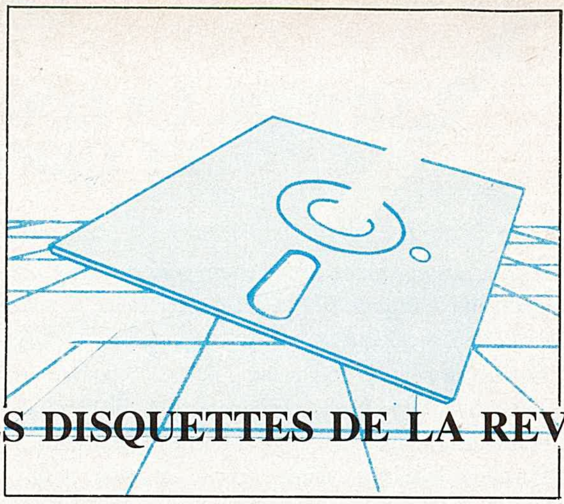
..........................................................

**Code Postal** .............................................. **Ville**

**Téléphone** ..........................................................

**Veuillez m'envoyer ..... logiciel(s) COUIC au prix unitaire de 340 F (TTC).**

**Ci-joint un chèque de ......... F. En règlement à Softin Communications.**



# LES DISQUETTES DE LA REVUE

DISQUETTES THÉMATIQUES

| N° | Revues | |
|---|---|---|
| 1 | 10,11 | Simulateur du fonct. d'un micro |
| 2 | 21 à 25 | Debug système et assembleur |
| 3 | 17 | Séquentiel indexé en BASIC |
| 4 | 19 à 23 | Éditeur et DAO en BASIC |
| 5 | 24 à 27 | Étude TURBO PASCAL (jeux, math.) |
| 6 | 28 | Conversion fichier DOS/WORDSTAR |
| 7 | 30, 32 | Étude C et PROLOG |
| 8 | 30 | Jeu SMILEY |
| 9 | 43, 47, 49 | PRSC, XPRINT et param. imprim. |
| 10 | 29, 30, 33, 34 | Imp. & graph., stylescript. |
| 11 | 26-28, 30-37, 38 | Adresses utiles + bib. souris |
| 12 | 52, 53 | Clavier intelligent et PASSWORD |
| 13 | 40, 49 | Scrolling texte + MENUDER |
| 14 | 50, 51 | Fenêtres Pascal + masques saisie |
| 15 | 32, 34, 40 | Hâchurage, Scrolling HGR, XFEN |
| 16 | 48, 54 | Courbes fractales |
| 17 | 39, 41, 42, 43 | Courbes et volumes en 3D |
| 18 | 34, 35 | MINITEL 1 |
| 19 | 36, 37 | MINITEL 2 |
| 20 | 40, 42 | MINITEL 3 |
| 21 | 50, 52, 53 | RT/CMOS, NEWAUX, INT 44H |
| 22 | 46, 48, 49 | MODIF/DEPL, SAUF, BATCH inter. |
| 23 | 38, 39, 41 | Résidents : ZIP5, TABLES, CONTEXT |
| 24 | 42, 43, 44 | Assemb. & DOS (ASM, HEURE, PRSC...) |
| 25 | 52, 53 | Gestion de fichier en TP4 |
| 26 | 40, 45, 54 | DEDICORE, SCAN & DEL, SAUVE |
| 27 | 44, 45, 40 | Étude TP. 4.0 + Fenêtrage HGR |
| 28 | 48 à 54 | Dossier TURBO C |
| 29 | 50, 54 | SECUDOS et prog. invisible |
| 30 | 51, 52, 53, 55 | DAO en TURBO PASCAL |
| 31 | 55, 56 | DBASE |
| 32 | 56, 58, 59, 60... | GENECR : Générateur d'écran en T.P. |
| 33 | 58, 60 | Compil. interp. et désas. PASCAL |

UTILITAIRES

| N° | Revues | Titres |
|---|---|---|
| 1 | 40 | DEDICORE : Tri sélectif des répertoires pour copier, effacer, renommer |
| 2 | 41 | CONTEXT : Un résident pour connaître votre configuration hard & soft |
| 3 | 43 | PRSC : Un résident pour rediriger et paramétrer la recopie d'écran |
| 4 | 45 | SCAN & DEL : Pour sélectionner et effacer vos fichiers inutiles |
| 5 | 47 | XPRINT : Un programme résident pour configurer l'imprimante |
| 6 | 49 | PARAMIMP : Un résident pour contrôler et utiliser votre imprimante |
| 7 | 54 | LS : Une commande DIR style UNIX |
| 8 | 57 | VDISK : Un disque virtuel dynamique |
| 9 | 57 | COPYDIR : Pour copier vos répertoires sur disquettes |
| 10 | 56 | RESIZE : Optimisez la taille de l'environnement de votre PC |
| 11 | 58 | SOSFILE : Si vous voulez récupérer des fichiers endommagés, cet utilitaire est pour vous |
| 12 | 58 | PCTEST : Pour tout connaître sur la configuration de votre machine |
| 13 | 56, 57, 59 | EXFILE : Analyseur de fichiers PASCAL |
| 14 | 58, 60 | HARDCOPY : Impression graphique d'écran EGA-VGA en N. & B. et nuances de gris |
| 15 | 61 | EDITPAL : Éditeur de palette pour EGA/VGA |
| 16 | 60 | XMEM : Utilisation de la mémoire étendue sous DOS |
| 17 | 64 | ANTIVIRUS : Traitement des virus PingPong et Jérusalem (sources et programmes) |
| 18 | 70, 71, 72 | XMS : OptimXMS : Optimiser la mémoire XMS |

---

**BON DE COMMANDE :**
**à adresser à SOFTIN COMMUNICATIONS**
**2 rue d'Amsterdam**
**75009 Paris - Tél. : 42.81.54.27**

Nom ..........................................................................................

Adresse ......................................................................................

Code Postal ........................................................ **Ville** ........................................

Téléphone ....................................................................................

Veuillez m'envoyer la ou les disquette (s) numéros : **THÉMATIQUES (160,00 F) :** ..........................
**UTILITAIRES (90,00 F) :** ..........................

Veuillez trouver ci-joint un chèque de........................ **× 160,00 F soit........................ Francs.**
........................ **× 90,00 F soit........................ Francs.**
**Total :..................... Francs.**

**en règlement à l'ordre de Softin Communications.**

**Date :** **Signature :**

Vous remarquerez que nous n'avons pas mis le mot ''virtual'' après la définition du constructeur ''init''. C'est tout à fait normal : le constructeur est toujours redéfini dans des objets irritants. Le mot ''virtual'' est donc implicite avec les constructeurs. Nous vous conseillons d'utiliser le mot ''virtual'' pour toutes les procédures. Lorsque votre librairie object sera définitivement arrêtée, vous pourrez enlever les mots virtual qui ne sont pas nécessaires.

Il existe deux formes d'héritage : l'héritage simple et l'héritage multiple. Le C++ accepte les deux formes, mais le Pascal n'accepte que la première forme. L'héritage simple a l'avantage d'être de conception plus facile.

## Le polymorphisme

La déclaration de procédures avec le mot ''virtual'' permettent de mettre en place un très puissant outil : le POLYMORPHISME (qui signifie ''plusieurs formes'' en grec).

Il consiste à donner un même nom à une action qui est commune à une hiérarchie d'objets, chaque objet redéfinissant son effet réel afin de l'adapter à ses besoins.

Dans l'exemple de l'objet ''personne'' et de ses deux héritiers (''client'' et ''salarié''), les trois procédures (''init'', ''affiche'' et ''done'' sont présentes. Elles font pourtant des actions différentes.

---


**BON DE COMMANDE**

à retourner
à la Revue de l'Utilisateur PC
2, rue d'Amsterdam
75009 Paris

Nom :
Société :
Adresse :
Code postal :
Ville :
Tél. :
Montant : 21 F × =
24 F × =
(à partir du n° 45)
Total achat revues =

\+ Frais Poste =

Montant du chèque =

**POUR COMMANDER LES ANCIENS NUMEROS**

Cochez les chiffres qui vous intéressent :

| | | | | |
|---|---|---|---|---|
| ☐ 1 | ☐ 2 | ☐ 3 | ☐ 4 | ☐ 5 |
| ☐ 6 | ☐ 7 | ☐ 8 | ☐ 9 | ☐ 10 |
| ☐ 11 | ☐ 12 | ☐ 13 | ☐ 14 | ☐ 15 |
| ☐ 16 | ☐ 17 | ☐ 18 | ☐ 19 | ☐ 20 |
| ☐ 21 | ☐ 22 | ☐ 23 | ☐ 24 | ☐ 25 |
| ☐ 26 | ☐ 27 | ☐ 28 | ☐ 29 | ☐ 30 |
| ☐ 31 | ☐ 32 | ☐ 33 | ☐ 34 | ☐ 35 |
| ☐ 36 | ☐ 37 | ☐ 38 | ☐ 39 | ☐ 40 |
| ☐ 41 | ☐ 42 | ☐ 43 | ☐ 44 | ☐ 45 |
| ☐ 46 | ☐ 47 | ☐ 48 | ☐ 49 | ☐ 50 |
| ☐ 51 | ☐ 52 | ☐ 53 | ☐ 54 | ☐ 55 |
| ☐ 56 | ☐ 57 | ☐ 58 | ☐ 59 | ☐ 60 |
| ☐ 61 | ☐ 62 | ☐ 63 | ☐ 64 | ☐ 65 |
| ☐ 66 | ☐ 67 | ☐ 68 | ☐ 69 | ☐ 70 |
| ☐ 71 | ☐ 72 | | | |

N° 9 - 17 - 18 - 19 - 20 - 22 - 23 - 34 - 35 - 36 - 37 épuisés

**Règlement par chèque bancaire :**
**à l'ordre de SOFTin Communications : 2, rue d'Amsterdam, 75009 Paris**
**Frais poste pour 1 n° 7,50 F - 2 n° 14,00 F - 3/4 n° 18,00 F - 5/6 n° 25,00 F**
**(au-dessus par multiple)**

personne.init : initialise nom et prénom.

client.init : initialise nom, prénom et montant.

employe.init : initialise nom, prénom, matricule et secu_soc.

Pourtant lorsque vous appellerez ces méthodes, vous écrirez : figure 11.

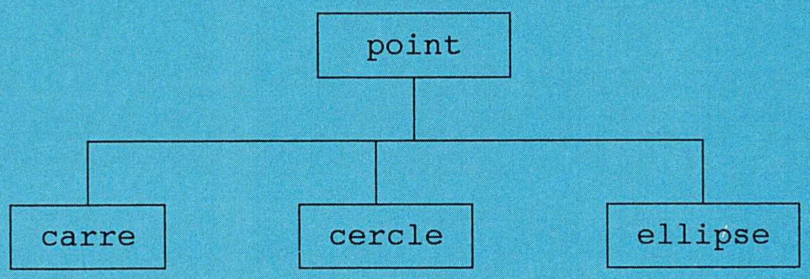
```
figure 11 : appel de procédures polymorphes

var
  pers  : personne;
  cli   : client;
  emp   : employe;

begin
  pers.init('Montbazet','François');
  cli.init('Dupont','Jean',3,9870.00);
  emp.init('Martin','Pierre',32000,'1600175101123');

  pers.affiche;
  cli.affiche;
  emp.affiche;

  pers.done;
  cli.done;
  emp.done;
end;
```

Voici un autre exemple donné par BORLAND durant la journée consacrée à la programmation orientée object : figure 12.

figure 12 : Exemple de l'utilité du polymorphisme.

| PASCAL traditionnel | PASCAL orienté objet |
|---|---|
| `case forme of`<br>`  point   : TracePoint;`<br>`  cercle  : TraceCercle;`<br>`  carre   : TraceCarre;`<br>`  ellipse : TraceEllipse;`<br>`else`<br>`  writeln('Forme inconnue');`<br>`end;` | `UneForme.trace;` |

hiérarchie de ces objets :



Voici donc les trois règles de la Programmation Orientée Objet. Vous aurez sans doute un peu de mal à programmer directement en objet. Ce que vous pouvez faire dans un premier temps, c'est de déclarer certaines structures comme des objets, et certaines autres de manière traditionnelle.

Mais à chaque fois que vous déclarerez une structure, posez-vous les questions : qu'est-ce qui caractérise ma structure ?, et est-ce qu'il y a des procédures qui ne peuvent "marcher" qu'avec cette structure ? Ainsi vous aurez fait le premier pas vers la Programmation Orientée Objet : l'encapsulation.

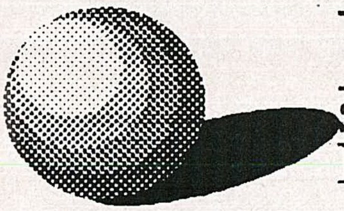
Vous retrouverez les exemples dans le listing fourni avec cet article.

Au travail...


OFFRE DE LANCEMENT

jusqu'au 31/03/92

le SHAREWARE du PROFESSIONNEL



PRO. Shareware

SARL au capital de 50000 F

BP 11 71120 CHAROLLES

1 disquette GRATUITE pour chaque commande

Procurez-vous de nombreux outils de programmations dans de nombreux langages, ainsi que des utilitaires et jeux du DOMAINE PUBLIC ou SHAREWARE internationaux au prix de 30 frs la disquette

envoyer à PRO.SHAREWARE BP 11 71120 CHAROLLES

Demande de CATALOGUE

Nom :  Prénom :

Adresse :

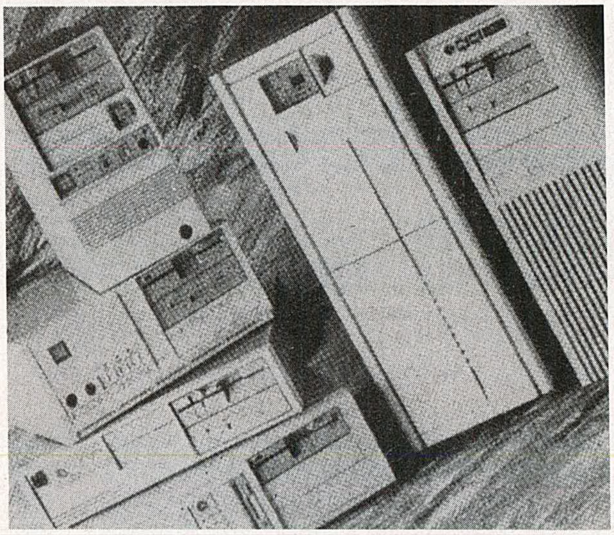
Code Postal :  Ville :

PC0292

```
uses crt;

  type

    ptr_personne = ^personne;


     personne = object
                   nom    : string[20];
                   prenom : string[12];

                   constructor init(xnom, xprenom : string);
                   procedure affiche;   virtual;
                   destructor done;     virtual;
                end;



     client = object(personne)
                   numero  : integer;
                   montant : real;

                   constructor init(xnom, xprenom : string;
                                    xnumero       : integer;
                                    xmontant      : real);
                   procedure affiche;   virtual;
                end;


        employe = object(personne)
                     matricule : integer;
                     secu_soc  : string[11];

                     constructor init(xnom,xprenom : string;
                                      xmatricule   : integer;
                                      xsecu_soc    : string);
                     procedure affiche;  virtual;
                  end;


 constructor personne.init(xnom, xprenom : string);
 begin
    nom    := xnom;
    prenom := xprenom;
 end;

 procedure personne.affiche;
 begin
    writeln;
    writeln('objet personne');
    writeln('nom    : ',nom);
    writeln('prénom : ',prenom);
    writeln;
 end;

 destructor personne.done;
 begin
 end;

 constructor client.init(xnom, xprenom : string;
                         xnumero       : integer;
                         xmontant      : real);
 begin
    personne.init(xnom,xprenom);
    numero  := xnumero;
    montant := xmontant;
 end;

 procedure client.affiche;
 begin
    writeln;
    writeln('objet client');
    writeln('numero : ',numero);
    personne.affiche;
    writeln('dettes : ',montant);
    writeln;
 end;
 constructor employe.init(xnom, xprenom : string;
                          xmatricule    : integer;
                          xsecu_soc     : string);
 begin
    personne.init(xnom,xprenom);
    matricule := xmatricule;
    secu_soc  := xsecu_soc;
 end;

 procedure employe.affiche;
 begin
    writeln;
    writeln('objet employe');
    writeln('matricule : ',matricule);
    personne.affiche;
    writeln('code secu : ',secu_soc);
    writeln;
 end;
var
  pers1 : ^personne;
  pers2 : ptr_personne;
  cli   : client;
  emp   : employe;
begin
  clrscr;
  new(pers1);
  pers1^.init('Montbazet','François');
  pers2 := new(ptr_personne,init('Martin','Pierre'));
  cli.init('Dupont','Jean',3,9870.00);
  emp.init('Martin','Pierre',32000,'1600175101123');
   pers1^.affiche;
   pers2^.affiche;
   cli.affiche;
   emp.affiche;

   pers1^.done;
   dispose(pers1);
   dispose(pers2,done);
   cli.done;
   emp.done;

 end.
```

# Que votre volonté soit faite !

## Votre micro-ordinateur sur mesure.



Solidement implantée dans le domaine de la fabrication de micro-ordinateurs depuis plus de quatre années, **CB Informatique** vous propose une configuration adaptée à vos besoins demeurant parfaitement évolutive au fil du temps.

Testés durant 72 heures dans nos laboratoires, les micro-ordinateurs CB vous assurent de la qualité professionnelle.

**Le coût ?** Nous vous en laissons seuls juges !

Voici donc quelques exemples de micro-ordinateurs CB standards pour lesquels nous nous ferons un plaisir de vous proposer un devis correspondant à vos choix.

| *Tous nos prix s'entendent H.T. départ CB.* | Hercules Monochrome | V.G.A. Monochrome | V.G.A. Couleur 800 x 600 |
|---|---|---|---|
| **CB 286 - 12** Disque dur 40 Mo AT Bus | **5 700** | **6 200** | **7 600** |
| **CB 386 SX - 20** Disque dur 80 Mo AT Bus | **11 200** | **11 650** | **13 900** |
| **CB 386 - 33** Disque dur 110 Mo AT Bus | **16 100** | **16 500** | **19 900** |

Toutes ces configurations comprennent : un boîtier 'Desktop', 1 mégaoctet de mémoire centrale, un lecteur 5,25"(1,2 Mo) ou 3,5" (1,44 Mo), les contrôleurs d'écran et de disques, un clavier 102 touches AZERTY ainsi que deux ports série et un port parallèle.

**Tarif spécial Revendeurs : nous consulter au tél : (1) 43.96.93.18 ou Fax : (1)43.96.07.44**

✂-----------------------------------------------------------------------------------

Je souhaite recevoir une proposition concernant le matériel suivant:

| Boîtier | Carte mère | Mémoire | Lecteurs | Disque dur | Écran | Souris | MS DOS 4.01 | Autres |
|---|---|---|---|---|---|---|---|---|
| *Desktop Mini-Tower Tower* | *286-12, 386 SX-20 386-20, 386-25 386-33* | *1, 2, 4, 8 Mo* | *1 Lecteur 5,25 ou 3,5 2 Lecteurs 5,25 ou 3,5* | *20,40,80,110 Mo ou supérieur* | *Hercules mono VGA mono VGA coul. 800x600* | *Oui Non* | *Oui Non* | *Par exemple : streamer, imprimante, scanner...* |
| | | | | | | | | |

Société : ______________________

Nom et prénom : ______________________

Adresse : ______________________

Code postal : __________

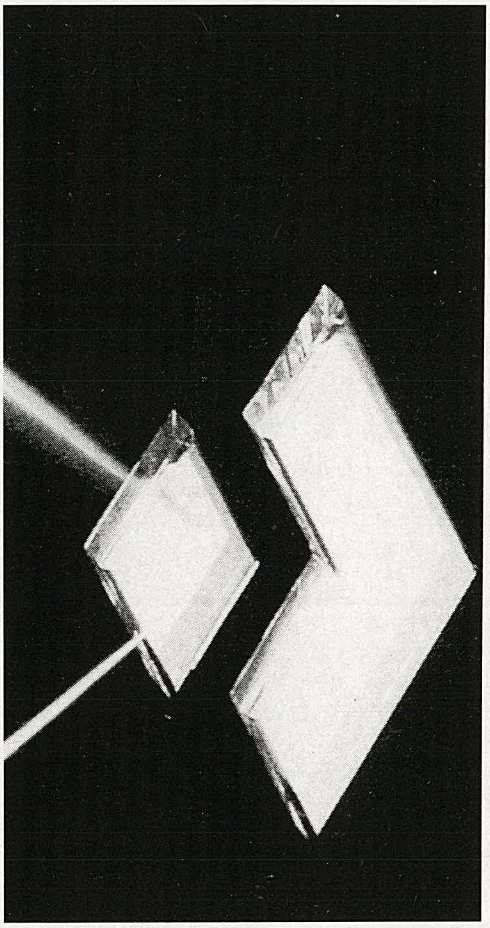
Ville : ______________________

Téléphone / Fax : ____________ / ____________

CB informatique

30, avenue du Maréchal de Lattre de Tassigny
**94410 SAINT MAURICE**
Tél : (1) 43.96.93.18
Fax : (1) 43.96.07.44

# La logique à 3 entrées



**La logique constitue une des disciplines de base de l'informatique : Algèbre de Boole, instructions logiques des microprocesseurs, ou fonctions logiques des tableurs et des langages de programmation... témoignent de son caractère à la fois omniprésent et prépondérant.**

Paul FRANCESCHI

Dans le jeu d'instructions des microprocesseurs, les instructions logiques occupent en effet une place de choix ; les trois principales sont bien connues des programmeurs en Assembleur.

— OR : OU
— XOR : OU EXCLUSIF
— AND : ET

Basées sur la logique à 2 entrées, c'est-à-dire qu'elles admettent en entrée deux opérandes, ces instructions délivrent ensuite en sortie un résultat logique.

En termes de langages procéduraux, il s'agit de fonctions logiques, qui comportent deux paramètres, et obéissent à la syntaxer suivante.

INT ORA (INT I, INT J)
INT XOR (INT I, INT J)
INT AND (INT I, INT J)

Ces fonctions et instructions familières possèdent en fait une caractéristique commune : elles appartiennent à la famille de la logique à 2 entrées dans laquelle une opération logique est effectuée sur les deux valeurs d'entrée, et un résultat est délivré, qui constitue la valeur de sortie.

Pour universelle et polyvalente qu'elle soit, la logique à 2 entrées n'en demeure pas moins — par définition — limitée et restreinte à deux entrées seulement. Comme nous le découvrirons à l'analyse, elle constitue en fait un développement partiel, embryonnaire et tronqué, d'une forme de logique aux potentialités infiniment plus larges et étendues : la logique à 3 entrées.

Cette dernière, comme nous le constaterons plus loin, contient et englobe la logique à 2 entrées dans sa totalité, mais offre en outre plusieurs centaines d'instructions logiques supplémentaires : au total, elle autorise 256 opérations logiques.

Leur nombre est si important qu'il nous sera impossible de les étudier individuellement, de manière exhaustive ; cependant nous analyserons de manière détaillée les principales d'entre elles.

Notre choix s'est porté sur quelques-unes des plus significatives, mais il demeure toutefois une sélection arbitraire, susceptible d'être modifiée ou complétée.

Les applications de la logique à 3 entrées sont universelles, et concernent des domaines aussi divers que le traitement graphique, le cryptage, le décryptage, la comparaison de fichiers, etc...

Le présent article se propose de définir tout d'abord les fécondes bases théoriques de cette nouvelle forme de logique, puis d'en présenter ensuite les applications pratiques, ainsi qu'un émulateur logiciel écrit en Langage Assembleur, qui vous permettra de l'utiliser dans vos programmes.

## Présentation : les concepts spécifiques

Avant d'aborder les opérateurs logiques à 3 entrées, il convient d'effectuer un retour sur la logique classique à 2 entrées — et même à 1 entrée — afin de parvenir à une meilleure compréhension des concepts nouveaux qui seront étudiés.

Par convention, nous utiliserons par la suite les lettres A B et C pour désigner respectivement les différentes entrées, dans chacune des logiques étudiées. La valeur de chaque entrée peut être 0 ou 1, ainsi que celle de chaque sortie.

Enfin, les numéros d'opérateurs logiques seront indiqués en hexadécimal, avec pour chacun d'entre eux, leur table de vérité.

## La logique à 1 entrée

Il s'agit là du type le plus élémentaire de logique, qui comprend seulement 4 opérateurs : une seule entrée, et une seule sortie pour ce mode particulier. La table de vérité qui lui correspond est la suivante.

| Entrée | Numéro d' Opérateur logique ( en héxadécimal) | | | |
|---|---|---|---|---|
| A | 00 | 01 | 02 | 03 |
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |

Quatre fonctions seulement existent dans ce type de logique ; on constatera la présence de l'opérateur de négation NOT, également présent dans le jeu d'instructions des microprocesseurs.

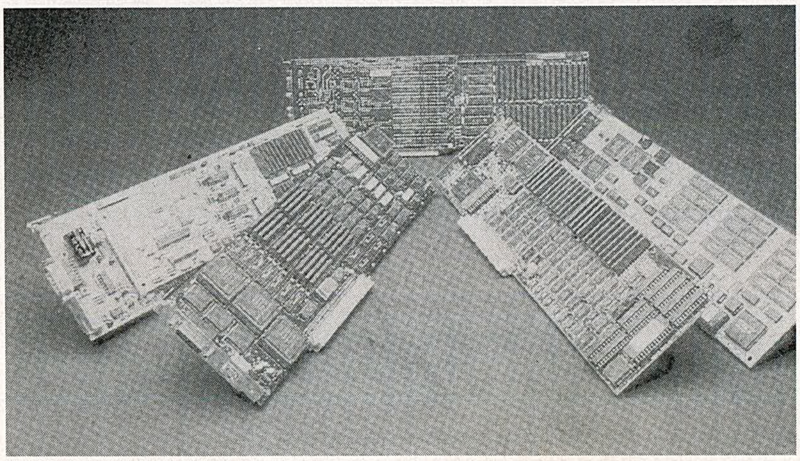
De même, et de manière plus inattendue, l'opérateur Copy n'est autre que l'instruction fondamentale MOV de l'Assembleur, qui — considérée comme une instruction logique — recopie simplement le contenu d'une entrée vers une sortie.

| Opérateur | Nom | Fonction |
|---|---|---|
| 00 | Nul | Mise à zéro systématique du résultat logique, quelle que soit l'entrée A |
| 01 | Copy | Recopie l' état de l'entrée A |
| 02 | Not | Négation. Réalise l'inversion de l' entrée A |
| 03 | Set | Mise à 1 systématique du résultat logique, quelle que soit la valeur d'entrée |

## La logique à 2 entrées

Elle comprend au total 16 opérateurs, correspondant à la table de vérité ci-dessous, en fonction des valeurs 0 ou 1 attribuées aux entrées A et B.

| Entrées | | Numéro d' Opérateur logique (en héxadécimal) | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |



| Opérateur | Nom | Fonction | Utilisation |
|---|---|---|---|
| 00 | Nul | Instruction de la logique à 1 entrée. Mise à zéro systématique du résultat logique, quelles que soient les entrées A et B | |
| 01 | And | Il s'agit du ET logique bien connu, qui ne délivre 1 que lorsque les deux entrées sont au niveau logique 1 | |
| 02 | Greater | Délivre 1 dans l' hypothèse unique où A est égal à 1 et B est égal à 0, c'est-à-dire lorsque A est strictement supérieur à B (A>B) | |
| 03 | Copy | Instruction de la logique à 1 entrée. Recopie l'entrée A | |
| 04 | Less | Délivre 1 dans le cas unique où A est inférieur à B, c'est-à-dire quand A est égal à 0 et B à 1 (A<B) | Voici un des opérateurs logiques à 2 entrées négligés par l'usage courant, mais qui s'avère d'une remarquable utilité, notamment pour réaliser des comparaisons |
| 05 | Copy_B | Recopie l'entrée B dans la valeur de sortie | |
| 06 | Xor | Réalise un classique OU EXCLUSIF | |
| 07 | Ora | Permet d'effectuer un OU logique traditionnel | |
| 08 | Non-Ora | Instruction logique NON-OU, qui est l'inverse de l'instruction OU | |
| 09 | Non-Xor<br>Equal | Inverse du OU EXCLUSIF. On peut également appeler cet opérateur Equal, car il délivre 1 quand les entrées A et B sont identiques | Cette fonction délaissée par l'usage habituel s'évère d'une remarquable utilité pour réaliser des comparaisons |
| 0A | Neg_B | Réalise l'inversion de l' entrée B | |
| 0B | Greater_Or_Equal | Délivre 1 quand A est supérieur ou égal à B (A>=B) | |
| 0C | Not | Instruction de la logique à 1 entrée. Réalise l'inversion de l' entrée A | |
| 0D | Less_Or_Equal | Délivre 1 lorsque A est inférieur ou égal à B (A<=B) | Autre instruction négligée |
| 0E | Non-And | Inverse de l'opérateur ET | |
| 0F | Set | Instruction de la logique à 1 entrée. Mise à la valeur logique 1 (vrai) du résultat logique | |

Cette analyse exhaustive illustre le fait que les instructions logiques habituellement utilisées (AND, OR, XOR...) ne constituent qu'un petit nombre parmi celles appartenant à cette famille.

## La logique à 3 entrées

Cette forme de logique aux riches possibilités contient non seulement les instructions de la logique à 1 entrée, mais également celles de la logique à 2 entrées, et possède en outre des dizaines d'instructions spécifiques.

Au total, elle comprend 256 instructions dont les principales sont commentées ici, avec leurs tables de vérité.

| Entrées | | | Numéro d' Opérateur logique à 3 entrées (en héxadécimal) | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| 00 | Nul | Mise à zéro systématique du résultat logique, quelles que soient les valeurs d' entrée. Possède son équivalent en logique à 1 et à 2 entrées |
| 01 | And_ABC | Réalise un ET logique entre les 3 entrées A,B et C; le résultat logique n'est vrai que lorsque les 3 entrées sont égales à 1 |
| 03 | And | ET logique à 2 entrées. L' état de l'entrée C est indifférent |
| 05 | And_AC | ET logique entre A et C |
| 0A | Less_AC | Délivre 1 dans le cas unique où A est inférieur à C |
| 0C | Greater | Opérateur de la logique à 2 entrées où A est supérieur strictement à C |
| 0F | Copy | Opérateur de recopie de l'entrée appartenant aux logiques à 1 et 2 entrées |

| A | B | C | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E | 1F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| 11 | And_BC | ET logique entre les entrées B et C |
| 14 | Different_AB_If_C_False_Set | Délivre 1 si A et B sont différents, sauf si C est nul, et dans cette hypothèse délivre 1 |
| 16 | XOR2_ABC | Instruction véritablement spécifique de la logique à 3 entrées: fournit 1 quand 2 entrées quelconques (mais pas 3) sont à 1; le cas où les 3 entrées sont vraies est exclu, de même que celui où une seule d'entre elles est vraie |
| 0A | Ora2_ABC | Instruction spécifique, qui délivre 1 quand 2 entrées au moins sont vraies; ce OU n'est pas exclusif, et par conséquent le cas où les 3 entrées sont vraies délivre aussi 1 |

| A | B | C | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E | 2F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| 22 | Greater_BC | Cas où B est strictement supérieur à C |
| 27 | Copy_If_C_False_Copy_B | Opérateur de copie conditionnel. Recopie A sauf si C est nul, et dans ce cas recopie B |
| 26 | Different_AB_If_C_True_Nul | Délivre 1 si A et B sont différents, sauf si C est vrai, et dans ce cas fournit 0 en sortie |

| A | B | C | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E | 3F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| 30 | Less | Logique à 2 entrées |
| 33 | Copy_B | Recopie l'entrée B |
| 3C | Xor | OU exclusif de la logique à 2 entrées |
| 3F | Ora | OU de la logique à 2 entrées |

| A | B | C | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| 41 | Equal_AB_If_C_False_Nul | Délivre 1 quand A et B sont égaux, sauf si C est nul, et dans ce cas délivre 0 |
| 44 | Less_BC | Teste l'infériorité de B par rapport à C |
| 4B | Equal_AB_If_C_False_Copy | Fournit 1 en sortie si A et B sont identiques, sauf si C est nul, et alors recopie l'entrée A |

| A | B | C | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 5A | 5B | 5C | 5D | 5E | 5F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| 50 | Not_If_C_False_Nul | Inverse l'entrée A sauf si C est nul, et dans ce das délivre 0 |
| 55 | Copy_C | Recopie l'entrée C |
| 5A | Xor_AC | OU exclusif entre A et C |
| 5F | Ora_AC | OU entre A et C |

| A | B | C | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 6A | 6B | 6C | 6D | 6E | 6F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| 63 | Equal_AB_If_C_False_Copy_B | Fournit 1 si A et B sont égaux, sauf si C est nul où à est délivré |
| 66 | Xor_BC | OU exclusif entre B et C |

| | | |
|---|---|---|
| 68 | Xor_ABC | Réalise un OU EXCLUSIF, mais entre les 3 entrées; le résultat est vrai lorsqu'une seule des 3 entrées est égale à 1 |

| A | B | C | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| 77 | Ora_BC | OU entre B et C |
| 7E | Different_ABC | Fournit le résultat logique 1 quand les 3 entrées sont différentes |
| 7F | Ora_ABC | Effectue un OU entre les 3 entrées; délivre 1 quand une des entrées au moins est égale à 1 |

| A | B | C | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| A | B | C | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| 97 | Non_Xor_ABC | Non-OU EXCLUSIF entre A,B et C |
| 99 | Non_Xor_BC | Non-OU EXCLUSIF entre B et C |

| A | B | C | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | AA | AB | AC | Ad | AE | AF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| A0 | Non-Ora_AC | Non-OU entre A et C |
| A5 | Non_Xor_AC | Non-OU exlusif entre A et C |
| AA | Not_C | Inverse l'entrée C |

| A | B | C | B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 | BA | BB | BC | Bd | BE | BF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| BB | Greater_BC | Cas où B est plus grand que C |
| BE | Different_AB_If_C_False_Set | Délivre 1 quand A et B sont différents, sauf si C est nul, où la valeur 1 est alors fournie |

| A | B | C | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | Cd | CE | CF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| C0 | Non_Ora | Non-OU entre A et B |
| CC | Neg_B | Inverse B |

| A | B | C | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | Da | Db | Dc | Dd | De | DF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| D8 | Not_If_C_False_Neg_B | Inverse A si B est vrai, ou inverse B si B est faux |

| A | B | C | E0 | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | EA | EB | EC | Ed | EE | EF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| EB | Equal_AB_If_C_False_Set | Délivre 1 si A et B sont égaux, sauf si C est nul où 1 est systématiquement délivré |

| A | B | C | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | FA | FB | FC | FD | FE | FF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Opérateur | Nom | Fonction |
|---|---|---|
| F0 | Not | Inversion de A (logique à 1 et 2 entrées) |
| FA | Not_If_C_False_Set | Inverse A sauf si C est nul, où la valeur 1 est fournie systématiquement |
| FF | Set | Résultat toujours vrai (logique à 1 et 2 entrées) |

D'une manière synthétique, on peut observer que la logique à 3 entrées présente un certain nombre de familles d'instructions, calquées sur les modèles généraux ci-dessous.

| Schéma général | Exemples |
|---|---|
| <opérateur logique à 1 entrée> | Nul, Copy, Not, Set . Ces 4 opérateurs sont présents dans la logique à 3 entrées |
| <opérateur logique à 2 entrées> | Greater,Less,Equal, And,Ora,Xor, Non-And, etc... La totalité des 16 opérateurs est incluse |
| - <opérateur logique à 2 entrées><condition><opérateur logique à 1 entrée> où la condition est exprimée en fonction de l' entrée C par exemple, qui peut être égale à 0 (False) ou 1 (True) | And_If_C_False_Copy<br>Xor_If_C_False_Set<br>Less_If_C_False_Nul<br>Equal_If_C_False_Not |

Il est clair que chacun des opérateurs à 3 entrées aurait pu à lui seul faire l'objet d'une étude ; toutefois l'analyse exhaustive des 256 types d'instructions aurait largement dépassé le cadre de la présente investigation.

Dans la seconde partie de cet article, nous nous attacherons à étudier les applications pratiques multiples de la logique à 3 entrées, en fournissant notamment en module en Assembleur permettant d'émuler cette dernière.

## Les applications

Les applications pratiques de la logique à 3 entrées sont universelles, et concernent des disciplines extrêmement variées ; nous étudierons un certain nombre d'entre elles, qui ont trait à des domaines tels que le cryptage, le traitement graphique, ou l'analyse statistique.

## Le cryptage

Les techniques de cryptage basés sur les opérations de la logique à 2 ou 3 entrées sont nombreuses : la plupart des opérations logiques s'analysent en effet comme une méthode particulière pour coder un texte, un programme, ou une image graphique.

Dans tous ces cas, l'algorithme de cryptage est très simple, puisqu'il fait appel à une des 256 instructions de la logique à 3 entrées ; pour le décryptage du texte, on utilisera la plupart du temps la même instruction que pour le cryptage. Le principe de base utilisé dans ces techniques, est de coder le fichier, l'image ou le texte, octet par octet, à l'aide d'une fonction logique, et de le restituer ensuite avec la même opération logique.

La méthode la plus simple pour coder le texte, est celle qui fait appel à l'instruction logique d'inversion de bits : on inverse chacun des bits des octets à coder, au moyen de l'instruction logique NOT, qui possède les codes.

— 02 en logique à 1 entrée.

— 0C (hexadécimal) en logique à 2 entrées.

— F0 (hexadécimal) en logique à 3 entrées.

Un caractère Ascii ''A'', stocké sous la forme hexadécimale 40 ou bien 01000000 deviendra après codage par inversion de bits DF en hexadécimal ou 10111111 en binaire. Tous les caractères du texte étant codés par cette méthode, on utilisera également l'opérateur NOT pour réaliser le décodage, qui restituera les octets d'origine.

NOT est une fonction de la logique à 1 entrée, et demeure une méthode de codage primaire, mais dont le principe est utilisé par les fonctionnalités plus sophistiquées des logiques à 2 ou 3 entrées.

En logique à 2 entrées, nous utiliserons les techniques plus élaborées, et dès lors plus difficiles à déchiffrer de l'inversion conditionnelle : les bits des octets à coder seront inversés, seulement si telle condition est satisfaite ; nous disposons à cet égard des opérateurs suivants.

NOT__IF__B__SET__ELSE__COPY
NOT__IF__B__NUL__ELSE__COPY

Chacun de ces opérateurs permet de réaliser une inversion, qui dépend de l'état de l'entrée B. Pour cette dernière, on utilisera un masque de cryptage, en fonction duquel, le bit correspondant de l'entrée A sera soit recopié, soit inversé. En d'autres termes, et pour le premier opérateur NOT__IF__B__SET __ELSE__COPY la technique utilisée est la suivante :

— si le bit du masque est à 1, on inverse (NOT) le bit à coder ;

— si le bit du masque est à 0, on recopie (COPY) le bit à coder.

Pour notre exemple du caractère Ascii ''A'', et si l'on utilise un masque de EC en hexadécimal (11101100 en binaire), on obtiendra.

| | |
|---|---|
| Entrée A | 01000000 |
| Entrée B (masque) | 11101100 |
| Sortie | 10101100 |

Bien entendu, rien n'oblige à se contenter d'un masque élémentaire, constitué d'un seul octet : on pourra utiliser une séquence-masque par exemple, ou le masque est constitué par une séquence de longueur variable.

Dans cette hypothèse, on pourra considérer le masque comme un mot de passe. Dans l'exemple suivant, la séquence codée remplace le masque constitué d'un seul octet. La séquence-masque est IMAGE, alors que le mot à coder est SONDE.

| | | | | | |
|---|---|---|---|---|---|
| Entrée A | 01010011 (S) | 01001111 (O) | 01001110 (N) | 01000100 (D) | 01000101 (E) |
| Entrée B (masque) | 01001001 (I) | 01001101 (M) | 01000000 (A) | 01000111 (G) | 01000101 (E) |
| Sortie | **00011010** | **00000010** | **00001110** | **00000011** | **00111000** |

Il va de soi que pour le codage d'un texte, à la place du mot IMAGE ici, on fera appel à la répétition de la séquence-masque continue : IMAGE-IMAGEIMAGEIMAGE, etc...

L'opérateur emprunté ci-dessus à la logique à 2 entrées, pourra être remplacé avantageusement par une formation de la logique à 3 entrées, appartenant à la famille des inverseurs conditionnels.

NOT__IF < opération logique à 2 entrées entre B et C >

C'est-à-dire par exemple.

Dans cette logique, on pourra par exemple se servir d'un double masque, et faire porter les conditions sur chacun des deux masques. Les opérateurs de la logique à 3 entrées adaptés à ce type de cryptage sont construits sur le modèle suivant.

| | |
|---|---|
| NOT_IF_B_XOR_C_ELSE_COPY | Inverse l' entrée A, sauf si (B OU exclusif C) est vrai, et dans ce cas recopie A |
| NOT_IF_B_AND_C_ELSE_COPY | Inverse l' entrée A, sauf si (B ET C) est vrai, et alors recopie A |
| NOT_IF_B_OR_C_ELSE_COPY | Inverse l' entrée A, sauf si (B OU C) est vrai, et dans ce cas recopie A |
| NOT_IF_B_GREATER_C_ELSE_COPY | Inverse l' entrée A, sauf si (B > C) est vrai, et dans cette hypothèse recopie A |

Par notre exemple, nous utiliserons le premier opérateur à savoir.

NOT__IF__B__XOR__C__ELSE__COPY

Nous nous servirons de deux masques ou clés, qui seront IMAGE, et LUNE (il est important que les deux clés choisies n'aient pas la même longueur, car procéder ainsi équivaudrait alors à n'utiliser qu'une seule clé). Le cryptage s'opèrera de la manière suivante, pour le mot SONDE, placé au début de notre texte à crypter.

| | | | | | |
|---|---|---|---|---|---|
| Entrée A | 01010011 (S) | 01001111 (O) | 01001110 (N) | 01000100 (D) | 01000101 (E) |
| Entrée B (premierm asque ) | 01001001 (I) | 01001101 (M) | 01000000 (A) | 01000111 (G) | 01000101 (E) |
| Entrée C (second masque) | 01001100 (L) | 01010101 (U) | 01001110 (N) | 01000101 (E) | 01001100 (L) |
| Sortie | 01010110 | 01010111 | 01000000 | 01000110 | 01001100 |

Avec cet opérateur, le bit du texte à coder est inversé si le OU EXCLUSIF effectué entre les bits correspondants des masques 1 et 2 produit un résultat vrai ; dans le cas contraire, il est simplement recopié. Pour décrypter le texte ainsi codé, il suffira de connaître les deux mots-clés servant de masque, et d'effectuer l'opération logique inverse, afin de retrouver le texte original.

## Le décryptage

Le décryptage s'opère en logique à 3 entrées exactement de la manière que le cryptage ; il s'agit simplement de l'opération inverse : en pratique, on utilisera pour le décryptage, suivant la méthodologie décrite ci-dessus, le même code que celui qui a servi au cryptage des données.

L'intérêt d'une telle démarche est loin d'être négligeable : en effet, l'usage d'un tel algorithme permet de simplifier et donc d'optimiser considérablement le processus de cryptage-décryptage.

Il s'agit même d'un des rares cas de réciprocité d'une procédure, où la même routine assure à la fois une fonction spécifique (ici, le cryptage), et la fonction inverse (le décryptage).

On pourra appliquer cette méthode non seulement à des données de type textuel mais également à des données graphiques, s'il est nécessaire de coder des images ; le codage bit-à-bit utilisé est particulièrement adapté à ces dernières.

## Le traitement graphique

Un autre des domaines où la logique à 3 entrées trouve un champ d'application intéressant, est celui du traitement graphique : une image graphique de type bitmap est stockée et mémorisée sous forme de zone-mémoire, consistant en une suite d'octets.

A titre d'exemple, on pourra ainsi réaliser, au niveau d'une image graphique :

— l'inversion conditionnelle des bits qui la composent, grâce à l'utilisation d'opérateurs de la famille des inverseurs conditionnels (NOT__IF__C__SET, NOT__IF__C__NUL, etc...) ;

— des opérations logiques du type OR__ABC, XOR__ABC, ou bien AND__ABC, qui étendent les fonctions logiques usuelles à 2 entrées, aux opérations à 3 entrées, permettant ainsi d'utiliser des doubles masques, ou bien de faire porter les traitements sur deux images à la fois, ou même trois ;

— la recopie conditionnelle d'une image, autorisée par l'utilisation des fonctions du groupe des opérateurs de recopie conditionnels (COPY__IF__C__SET, COPY__IF__AND__AB, COPY__IF__XOR__AB, etc...).

## L'analyse logique statistique

Une des fonctionnalités les plus intéressantes fournies par la logique à 3 entrées, est celle qui permet une analyse de type statistique sur des données. Celles-ci, qu'elles soient graphiques, textuelles, ou numériques, sont toujours en définitive stockées sous forme de bits. Lorsqu'on désire en effet étudier et déterminer la nature de certaines données, il est particulièrement important de disposer d'un outil qui permette de calculer le nombre de bits qui les composent.

On peut ainsi amené à étudier des données, en souhaitant connaître :

— le nombre d'octets dont le bit 7 est à 1,

— le nombre d'octets dont le quartet haut est nul,

— le nombre d'octets dont les bits 5 et 2 sont nuls.

Des informations statistiques de cette nature revêtent une signification toute particulière, dans des applications de reconnaissance d'écriture, de signes ou de formes, ou bien lorsqu'on souhaite rechercher par exemple la méthode de cryptage d'un texte, etc...

C'est ainsi que pour déterminer le nombre d'octets dont le bit 7 est égal à 1 dans une zone de données, on utilisera un masque égal à 01111111 en binaire (7F en hexadécimal) en réalisant un ET logique (AND) sur la valeur d'entrée : si la valeur de sortie est égale à 11111111 (FF en hexadécimal), le bit 7 est bien à 1.

La logique à 3 entrées permettra de réaliser le même traitement sur deux zones-mémoire simultanément : on se servira toujours du masque 7F, mais on utilisera l'opérateur AND__ABC, qui délivre 1 lorsque les trois entrées ssont à 1. Si les bits 7 des deux zones sélectionnées sont à 1, le résultat en sortie est FF en hexadécimal.

On pourrait encore appliquer la logique à 3 entrées au cas de l'analyse précédente : au lieu d'utiliser l'opérateur à 2 entrées AND sur toute la zone-mémoire considérée, on peut se servir de l'opérateur à 3 entrées correspondant (AND__ABC), de façon à optimiser le traitement.

Si la zone-mémoire à analyser comporte 10 000 octets, on réalisera un gain en rapidité d'exécution d'un facteur 2, puisque l'analyse pourra porter simultanément sur deux zones consécutives de 5 000 octets chacune. On réalisera ainsi un AND__ABC entre :

— l'octet de rang 0,
— l'octet de rang 5 000,
— le masque FF.

A la seconde opération, le AND__ABC portera sur :

— l'octet de rang 1,
— l'octet de rang 5 001,
— le masque FF.

Et ainsi de suite...

En procédant ainsi, on divisera le temps de traitement par deux.

Cette méthodologie peut être généralisée à toutes les opérations logiques à 2 entrées portant sur des zones-mémoire continues, auxquelles on pourra appliquer un opérateur de la logique à 3 entrées portant sur les deux moitiés de la zone à traiter : dans tous les cas, on obtiendra une division par deux du temps de traitement.

Au terme de notre étude des applications permises par la logique à 3 entrées, nous devons garder présents à l'esprit que ces quelques exemples ne constituent que l'illustration des propriétés de plusieurs opérateurs choisis parmi 256, adaptés à certains domaines particuliers. En tout état de cause, il ne s'agit que d'une sélection, que l'on pourrait aisément élargir à d'autres domaines.

## Un émulateur en Assembleur

Le module ci-dessous, réalisé avec l'Assembleur MASM de Microsoft comprend un émulateur pour la logique à 3 entrées ; les fonctions logiques sont définies par une série d'instructions EQU. On fait ensuite appel au sous-programme principal LOG30, réalisant l'opération logique à 3 entrées, en utilisant la Macro LOG3.

```
PUBLIC  ENT0,ENT1,ENT2,LOG_TYP,L3B8RLT,LOG3BIT0,LOG30
;
RETN_DOS        MACRO
MOV     AL,0
MOV     AH,4CH
INT     21H
ENDM
;
INIT_11 MACRO           BYT0,BYT1
MOV     AL,BYT1
MOV     BYT0,AL
ENDM
;
; Macro réalisant l'opération logique à 3 entrées.
LOG3    MACRO           TYP,ENT00,ENT01,ENT02,RLT
INIT_11 LOG_TYP,TYP
INIT_11 ENT0,ENT00
```


# Bulletin d'abonnement pour 1 an

**10 numéros au prix de : France : 220 F T.T.C., Etranger : 300 F**
**Je souscris un abonnement d'un an à la Revue de L'utilisateur de l'IBM-PC**

Nom : ...........................................................................
Société : ........................................................................
Adresse : ........................................................................
.................................................................................
Code postal : ................ Ville : ................. Tél. : .................

Chèque bancaire joint à l'ordre de SOFtin Communications (France).

**A renvoyer à : La Revue de l'Utilisateur de l'IBM—PC, 2, rue d'Amsterdam, 75009 Paris.**

```
INIT_11  ENT1,ENT01
INIT_11  ENT2,ENT02
CALL     LOG30
MOV      RLT,AL
ENDM
;
; Les codes d'une sélection de
fonctions logiques à 3 entrées.
Nul              EQU     00H
And_ABC          EQU     01H
And              EQU     03H
And_AC EQU       05H
Less_AC EQU      0AH
Greater          EQU     0CH
Copy             EQU     0FH
And_BC EQU       11H
Xor2_ABC         EQU     16H
Ora2_ABC         EQU     1AH
Greater_BC       EQU     22H
Less             EQU     30H
Copy_B           EQU     33H
Xor              EQU     3CH
Ora              EQU     3FH
Less_BC EQU      44H
Copy_C           EQU     55H
Xor_AC           EQU     5AH
Ora_AC           EQU     5FH
Xor_ABCEQU       68H
Ora_ABCEQU       7FH
Non_Xor_ABC      EQU     97H
Non_Ora_AC       EQU     0A0H
Not_C            EQU     0AAH
Not_B            EQU     0CCH
Not              EQU     0F0H
Set              EQU     0FFH
;
;
DATA    SEGMENT         PUBLIC
;
O1O8ZON          DW      0
O8O1RES          DB      0
LOG_TYP          DB      0
L3B8SAVX         DW      0
L3B8ZON          DB      0,0,0,0,0,0,0,0
L3B8RLTDB        0
L3B8SAV          DB      0
ENT0             DB      0
ENT1             DB      0
ENT2             DB      0
LOG_HTP          DB      0,0,0,0,0,0,0,0
LOG_CAS          DW      0
RSV              DB      0
ENT00   DB       0
ENT10   DB       0
ENT20   DB       0
;
DATA    ENDS
;
;
```

```
CODE    SEGMENT     PUBLIC
ASSUME CS:CODE, DS:DATA, ES:DATA, SS:PILE
;
LOG3BIT0:
;
; Réalise une opération logique à 3 entrées au niveau d'un bit
; pour chaque entrée.
CALL    XPND_TYP0
CALL    GLUE_ENT0
MOV     DI,LOG_CAS
MOV     AL,LOG_HTP[DI]
RET
;
GLUE_ENT0:
; Compacte les trois bits d'entrée en un octet dans AL.
; Le résultat est aussi placé ensuite dans LOG_CAS (WORD).
;
MOV     AL,ENT2
MOV     RSV,0
SHR     AL,1
RCL     RSV,1
MOV     AL,ENT1
SHR     AL,1
RCL     RSV,1
MOV     AL,ENT0
SHR     AL,1
RCL     RSV,1
MOV     AL,RSV
MOV     AH,0
MOV     LOG_CAS,AX
RET
;
CVHEXBIN:
;
; Transforme un octet en 8 octets contenant soit 0 soit 1,
; en fonction de la valeur des bits de l'octet d'origine.
MOV     BX,O1O8ZON
CLC
MOV     DI,7
C1:
SHR     AL,1
JC      C2
MOV     [BX+DI],BYTE PTR 0
JMP     C3
C2:
MOV     [BX+DI],BYTE PTR 1
C3:
DEC     DI
CMP     DI,0FFFFH
JNE     C1
RET
;
CVBINHEX:
;
; Transforme 8 octets contenant 0 ou 1 en un seul octet.
MOV     BX,O1O8ZON
MOV     O8O1RES,0
MOV     DI,7
```

```
V1:
MOV     AL,[BX+DI]
AND     AL,1
JNE     V2
CLC
RCL     0801RES,1
JMP     V3
V2:
STC
RCL     0801RES,1
V3:
DEC     DI
CMP     DI,0FFFFH
JNE     V1
RET
;
XPND_TYP0:
;
MOV     AL,LOG_TYP
PUSH    AX
MOV     BX,OFFSET    LOG_HTP
MOV     0108ZON,BX
POP     AX
CALL    CVHEXBIN
RET
;
LOG30:
;
; La procédure qui réalise l'opération logique, à partir de:
; LOG_TYP: le code de l'opération logique à 3 entrées
; ENT0: la première entrée
; ENT1: la seconde entrée
; ENT2: la troisième entrée
; le résultat logique est placé dans AL, et L3B8RLT.
MOV     DI,0
L1:
MOV     L3B8SAVX,DI
CMP     DI,0
JE      L2
SHR     ENT0,1
SHR     ENT1,1
SHR     ENT2,1
L2:
CALL    LOG3BIT0
MOV     DI,L3B8SAVX
MOV     L3B8ZON[DI],AL
INC     DI
CMP     DI,8
JNE     L1
MOV     BX,OFFSET    L3B8ZON
MOV     0108ZON,BX
CALL    CVBINHEX
MOV     AL,0801RES
MOV     L3B8RLT,AL
RET
;
;
;
```

```
        DEBUT:
        ;
        MOV     AX,DATA
        MOV     DS,AX
        MOV     ES,AX
        ;
        ; Exemple d'appel du ET logique à 3 entrées (And_ABC), dont le résultat dans AL et
L3B8RLT est 55H.
        LOG3    And_ABC,01010101b,11111111b,11111111b,AL
        RETN_DOS
        ;
        CODE    ENDS
        ;
        PILE    SEGMENT     STACK
        DW      512 DUP(?)
        PILE ENDS
        ;
        END     DEBUT
        ;
        .
```

## Les implications Hardware

L'émulateur logiciel de la logique à 3 entrées présenté ci-dessus, est conçu comme un sous-programme, auquel une macro-instruction donne accès directement.

On peut penser cependant que des instructions de logique à 2 ou 3 entrées pourraient avec profit être directement intégrées au jeu d'instruction des microprocesseurs, pour les raisons suivantes.

— Les applications rendues accessibles directement par ce type de logique d'un degré supérieur, sont à la fois extrêmement puissantes et variées.

— La logique à 3 entrées émule complètement les instructions traditionnelles en langage-machine telles que AND, ORA, XOR, mais aussi NOT, et encore, comme nous l'avons montré dans la première partie de cet article, MOV. Une seule instruction pourrait ainsi en remplacer plusieurs, autorisant ainsi — dans l'esprit des processeurs à architecture RISC — une optimisation des performances.

— L'intégralité des instructions logiques à 2 ou 3 entrées serait ainsi incluse dans le jeu d'instruction du microprocesseur, à la place d'une sélection de ces dernières. On aurait ainsi 16 instructions logiques pour la logique à 2 entrées, et 256 instructions pour la logique à 3 entrées.

On disposerait ainsi de deux instructions logiques en langage-machine.

— LOG : réalise une instruction logique à 2 entrées, et répond à la syntaxe suivante :

LOG 〈constante/registre〉,〈registre/mémoire〉,〈registre/mémoire〉

Où le premier argument (constante/registre) contient le code (entre 0 et 15) de l'opération logique à effectuer, et les deux autres arguments sont les deux entrées, alors que le résultat en sortie est placé dans le second argument, conformément à l'usage actuel en langage-machine. Ainsi, une instruction telle que :

LOG 5, AX, BX

ou bien

__AND EQU 05H
LOG __AND, AX, BX

réalise une instruction de logique à 2 entrées de code 5, c'est-à-dire un ET logique entre les AX et BX, et place le résultat dans AX.

— LOG3 : réalise une instruction logique à 3 entrées, et obéit à la syntaxe :

LOG 〈constante/registre〉,〈registre/mémoire〉,〈registre/mémoire〉

Où le premier argument (constante/registre) contient le code (entre 0 et 255) de l'opération logique à effectuer, et les trois arguments suivants sont les trois entrées, alors que le résultat en sortie est placé dans le second argument. Ainsi, l'instruction suivante :

LOG3 1, AX, BX, DX

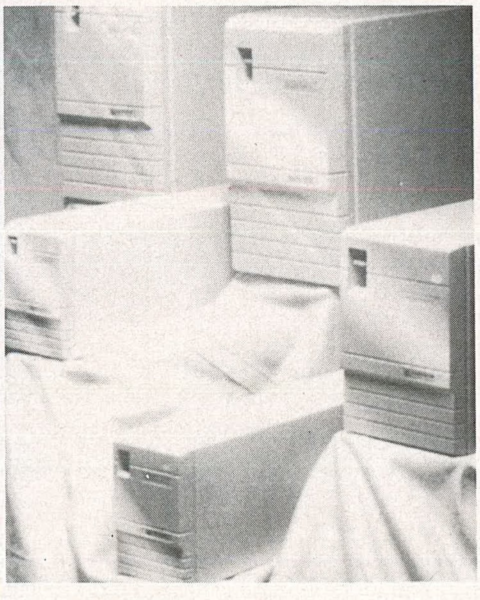
ou bien :

AND__ABC EQU 01H
LOG3 AND__ABC, AX, BX, DX

réalise une instruction de logique à 3 entrées de code 01H, c'est-à-dire un ET à 3 entrées entre AX, BX et DX, et place le résultat dans AX.

# PDEL : un utilitaire qui vous permet de protéger la confidentialité de certaines données en les détruisant de façon définitive

**Le système MS-DOS fait preuve, reconnaissons-le, d'une certaine clémence vis-à-vis de l'utilisateur distrait.**



Jean-Michel SPINER

Les conséquences potentiellement catastrophiques d'un Del involontaire peuvent en général être évitées.

En effet, si les fichiers détruits sont inaccessibles par les voies habituelles, la plupart des informations sont encore physiquement présentes sur le disque.

La destruction d'un fichier a deux conséquences : la première lettre du nom est remplacée par le caractère '?', et l'espace disque qui était attribué est maintenant considéré comme libre par le sytème pour y réaliser des écritures.

C'est pour cette raison que, si la récupération est possible, elle doit se faire avant que les informations n'aient été écrasées.

De nombreux utilitaires dont certains shareware permettent de restaurer les fichiers détruits comme si rien ne s'était passé.

Parmi les plus célèbres, citons les incontournables PCTools et Norton Utilities.

Les versions anciennes de PCTools ne permettent pas toujours une restauration parfaite, surtout sur les gros fichiers.

Si ce défaut a disparu dans les versions récentes (PCShell), les manipulations restent un peu fastidieuses.

On lui préférera qu (Quick Unerase), du grand maître Peter Norton, qui allie l'efficacité à une grande facilité d'emploi.

Dans les cas un peu plus délicats, lorsqu'une partie du fichier a été écrasée, le programme nu du même Norton vous permettra d'aller à la pêche aux clusters.

Les destructions réversibles, si elles apportent une certaine tranquillité d'esprit, ne sont pas toujours les bienvenues.

Elles vous empêchent en effet d'avoir la certitude qu'une fois détruites, des données confidentielles ne pourront pas être relues.

C'est là que qu'intervient PDEL, le Paranoïaque DELete.

Avant de détruire un fichier, il l'écrase avec un caractère paramétrable, rendant ainsi toute reconstruction absolument impossible.

L'utilisation de PDEL est très simple :

Pdel [ – cx] fichier(s)

Lorsque le flag c est activé, x représente le caractère de remplacement. Par défaut, PDEL utilisera le caractère '='.

Il est inutile de préciser que l'utilisation de PDEL est extrêmement dangereuse, et l'opérateur tournera sept fois son index autour du clavier avant de valider la commande.

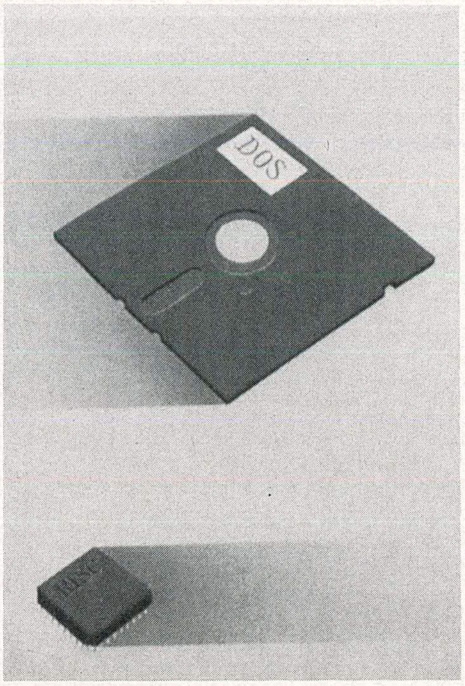
Pour atténuer les risques de mauvaises manipulations, vous pouvez rajouter des confirmations de destruction, soit pour chaque fichier, soit seulement dans le cas d'un ''pdel *.*''.

```
10/10/1991  01:14        FICHIER  :  PDEL.C                                PAGE  1
/*
        +------------------------------------------+---------+
        !                PDEL                      !  1.00   !
        +------------------------------------------+---------+
        !                                                    !
        !      ± FONCTION   :   Paranoaque Delete            !
        !                       Destruction de fichiers      !
        !                       avec crasement de la zone    !
        !                       de donnes                    !
        !                                                    !
        !      ± AUTEUR     :   Jean-Michel SPINER           !
        !                                                    !
        +----------------------------------------------------+
*/

#include <stdio.h>
#include <dos.h>

#define TRUE 1
#define FALSE 0
#define EXIT_OK 0
#define EXIT_ERR_SYNTAX 101
#define EXIT_FILE_NOT_FOUND 102
#define LONG_BUFFER 16000

unsigned char del_char = '#';
unsigned char buffer[LONG_BUFFER];
unsigned char path[80];

/*  ------------------------------------------------  */
/*                      MAIN                          */
/*  ------------------------------------------------  */
main(argc,argv)
int argc;
char *argv[];
{
   register int i1;
   struct find_t findfile;

   if (argc<2) {
      fprintf (stderr,"\nSyntaxe : PDEL [-cn] fichier(s)\n\n");
      exit (EXIT_ERR_SYNTAX);
   }
   for (i1=1;i1<argc-1;i1++) {
      if (argv[i1][0]=='-') {
         switch (argv[i1][1]) {
            case 'c' :
               del_char = argv[i1][2];
               break;
            default :
               fprintf (stderr,"\nErreur param : PDEL ");
               fprintf (stderr,"[-cn] fichier(s)\n\n");
```

```
10/10/1991  01:14        FICHIER  :  PDEL.C                                PAGE  2
               exit (EXIT_ERR_SYNTAX);
               break;
         }
      }
   }
   memset (buffer,del_char,LONG_BUFFER);
   strcpy (path,argv[argc-1]);
   if (_dos_findfirst(path,_A_NORMAL,&findfile)==0) {
      Pdel (findfile.name,findfile.size);
      while (_dos_findnext(&findfile)==0) {
         Pdel (findfile.name,findfile.size);
      }
   }
   else {
      fprintf (stderr,"\n%s non trouv\n\n",path);
      exit (EXIT_FILE_NOT_FOUND);
   }
   printf ("\n");
   exit (EXIT_OK);
}

/*                      PDEL                        */
/*--------------------------------------------------*/
Pdel (char *nom_fish,long size)
{
   register int i1;
   unsigned int nb_buffers;
   unsigned int reste;
   FILE *workfile;

   printf ("\n%s   ... ",nom_fish);
   if ((workfile=fopen(nom_fish,"wb"))==NULL) {
      printf ("Err ouverture");
      return (-1);
   }
   nb_buffers = (int)(size/LONG_BUFFER);
   reste = size % LONG_BUFFER;
   for (i1=0;i1<nb_buffers;i1++)
      fwrite(buffer,LONG_BUFFER,1,workfile);
   fwrite(buffer,reste,1,workfile);
   fclose (workfile);
   remove (nom_fish);
   printf ("Ok");
   return (0);
}
```

# Reconstruction d'une File Allocation Table

Jacques ROULLIER




**Récupérer un fichier ASCII ou un fichier de traitement de texte sur une disquette dont les Files Allocations Tables ont été effacées. (Peut-être à cause d'un cheval de Troie...) ou d'un formatage interrompu.**

## Pourquoi un nouveau programme

Les différents utilitaires du commerce sont capables de reconstruire un Boot Sector ou de retrouver le premier Cluster d'un fichier mais n'ont aucune possibilité de retrouver le secteur suivant quand les FAT ont été effacées.

Certains donnent la possibilité de créer un nouveau fichier pour chaque nouveau secteur, le DOS permettant ensuite de chaîner ces fichiers. Le moins qu'on puisse dire est que ce n'est pas pratique.

Le programme joint permet d'écrire des fichiers qui recevront autant de secteurs récupérés que l'utilisateur sera capable d'en reconnaître.

## Utilisation

Une disquette 5 1/4'' ou 3 1/2'' a perdu ses FAT. Mettez-la dans son lecteur et choisissez un autre lecteur (disquette ou disque dur) qui recevra les fichiers récupérés, créez un répertoire si besoin.

1) Le programme proposera d'éditer le Root Directory (ou les autres) soit à l'écran, soit sur imprimante (LPT1), cette liste donnera le numéro logique du premier secteur, le nombre d'octets et le nombre de secteurs du fichier.

Si le Root Directory a aussi été détruit il faudra procéder au hasard jusqu'à trouver le premier secteur non touché (par exemple par un formatage mal venu et interrompu à la volée, quelques pistes ont été perdues mais pas tout).

2) Utilisant le numéro du premier secteur réintroduit par l'utilisateur le programme affichera le contenu de ce secteur. Dans la version jointe les codes ASCII < 32 ne sont pas affichés pour éviter d'envoyer à l'écran des commandes hors sujet et le codage du Carriage Return par le programme de traitement CHIWRITER ('\') est utilisé pour envoyer les retours à la ligne. Cela est à modifier suivant le traitement de texte que vous utilisez et ses codages.

3) Reconnaissant le texte affiché l'utilisateur pourra lui redonner un nom qui aura automatiquement l'extension '.RVR' (pour ''Recovery'') et pourra le sauvegarder sur le nouveau support. Si le fichier existe déjà le nouveau secteur lui sera ajouté, sinon le fichier sera créé.

On peut alors lire, afficher et sauvegarder le secteur suivant ou tout autre secteur.

Il peut être bon de faire une liste manuelle des secteurs sauvegardés pour pouvoir reprendre le travail après la pause café. Et les secteurs à récupérer seront rarement contigus ce qui suppose recherche et patience, mais moins que pour tout retaper.

## Données techniques

La plupart viennent de la Revue de l'Utilisateur PC, éventuellement corrigées, mais sans éviter les redites pour avoir un texte complet, d'autres viennent de Micro-Systèmes et d'autres encore de droite, de gauche ou sont le résultat d'essais.

## Organisation des disquettes et disques durs

Secteur 0 : le Boot Sector commence par une table d'identification.

| OffSet | Long | Contenu | 5 1/4 | 3 1/2 | 20Mo | |
|---|---|---|---|---|---|---|
| 0h | 3 | Saut au programme de Boot (OffSet 036h) | EB 34 90 | | | h |
| 3 | 8 | Nom du programme de formattage | SANYO, IBM,... | | | |
| B | 2 | Nb d'Octets par Secteur | 512 | 512 | 512 | d |
| D | 1 | Nb de Secteurs par Cluster(*) | 2 | 2 | 4 | |
| E | 2 | Nb de Secteurs du programme de Boot | 1 | 1 | 1 | |
| 10 | 1 | Nb de FAT (File Allocation Table) | 2 | 2 | 2 | |
| 11 | 2 | Nbre maxi d'Entrées dans Root Directory | 112 | 112 | 512 | d |
| 13 | 2 | Nb total de Secteur du Disque | 720 | 1440 | 41463 | d |
| 15 | 1 | Contenu du premier octet de la FAT (**) | FD | F9 | F8 | h |
| 16 | 2 | Nb de Secteurs occupés par une FAT | 2 | 3 | 29 | h |
| 18 | 2 | Nb de Secteurs par Cylindre | 9 | 9 | 11 | h |
| 1A | 2 | Nb de Têtes de Lecture/Ecriture | 2 | 2 | 4 | |
| 1C | 2 | Nb de mauvais secteurs (dique dur) | | | | h |

1) Le DOS ne lit pas le disque Secteur par Secteur mais Cluster par Cluster.

Secteurs 1 à 4 ou 1 à 6 ou ... : FAT

Il y en a 2 copies identiques bien qu'on ne sache pas très bien quel programme utilise la deuxième !

Le Root Directory (voir plus loin) donne pour chaque fichier le numéro de l'Entrée dans la FAT et à partir de cette Entrée la FAT donne les suivantes et les numéros de secteurs correspondants.

Il s'en suit que l'espace minimum occupé par un fichier même très court est de 1 Cluster. Le choix du nombre de Secteurs par Cluster est un compromis entre la rapidité d'accès et la perte de place. Il n'est pas modifiable par l'utilisateur de MS-DOS, il l'est pour l'utilisateur du système d'exploitation Prologue.

Pour les disques durs et les disquettes 3 1/2'' chaque entrée occupe 2 octets, pour les disquettes 5 1/4'' elle n'occupe que 1.5 octet ce qui complique sérieusement les calculs !

2) Cet octet est appelé ''Media Descriptor'', il sert de contrôle du type de disque et permet la reconstruction de la Table d'Identification quand la FAT n'est pas effacée.

FAT typique d'un disque dur ou d'une disquette 3 1/2''.

| OffSet | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Contenu | F8 | FF | FF | FF | 03 | 00 | 04 | 00 | 05 | 00 | |
| Entry # | 0 | | 1 | | 2 | | 3 | | 4 | | ... |

FAT typique d'une disquette 5 1/4''.

| OffSet | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ... |
|---|---|---|---|---|---|---|---|---|---|---|
| Contenu | F9 | FF | FF | 03 | 40 | 00 | 05 | 60 | 00 | |
| N° Entrée | 0 | 1 | 2 | 3 | 4 | 5 | | | | ... |

OffSet 0 : Media Descriptor
1, 2 (et 3) : réservés
= Les Entrées 0 et 1 sont réservées

Valeur du Média Descriptor : (Micro-Systèmes Octobre 1989)

| | Nbre de Faces | Nbre de Secteurs par Cylindre | Nbre de Piste par Face |
|---|---|---|---|
| F9 | 2 | 9 | 80 |
| FA | 1 | 8 | 80 |
| FB | 2 | 8 | 80 |
| FC | 1 | 9 | 40 |
| FD | 2 | 9 | 40 |
| FE | 1 | 8 | 40 |
| FF | 2 | 8 | 40 |
| F8 | Disque dur | | |

Le contenu des octets des autres OffSets permet de trouver le N° d'Entrée suivant. Ce N° d'entrée permet de calculer le N° logique du Secteur où sont les données. Un contenu = 0000 indique un secteur libre.

Calculer le N° d'Entrée suivant :

Dans le premier cas (3 1/2″ ou disque dur) :

N° d'Entrée × 2 → FAT Offset → Word → N° suivant

Un Word = FFFFh indique la fin du fichier.

Dans le second cas, des calculs sont nécessaires (déjà publiés dans la Revue de l'Utilisateur PC).

| N° d'Entrée | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| N° d'Entrée × 1.5 | 3 | 4.5 | 6 | 7.5 |
| La partie entière-donne un OffSet | 3 | 4 | 6 | 7 |
| Cet Offset donne un Word | 4003 | 0040 | 6005 | 0060 |
| N° Entrée pair : masquer le MSB | 0003 | | 0005 | |
| N° impair : Shifter 4 bits vers la droite | | 0004 | | 0006 |
| N° d'Entrée suivant | 3 | 4 | 5 | 6 |

Les valeurs FF0h to FF6h sont réservées, toute valeur > FF7h indique la fin du fichier.

Calculer l'adresse logique

Les secteurs sont numérotés séquentiellement de 0 à 719 ou de 0 à 1 439 ou ... donnant un N° logique du secteur.

Nombre de Secteurs utilisés par le Système :

= Nbre de Secteurs de Boot + 2 × Nbre de Secteurs par FAT + Nbre de Secteurs du Root Directory = 12d pour une 5 1/4″ ou 14d pour une 3 1/2″ ou 115d pour un DD 20 Mb.

N° Logique du Secteur = (N° d'Entrée − 2) * Nbre de Secteurs par Cluster + Nbre de Secteurs pour le Système.

Secteurs 5 à B ou 7 à D ou ... : Root Directory

Chaque Entrée dans un Répertoire occupe 32d octets. (32 × 112 Entrées/512 = 7 secteurs).

| OffSet | Long | Contenu |
|---|---|---|
| 0h | 8 | Sous-Rep ou N.de F. } ou Nom de |
| 8 | 3 | Extension } Volume |
| B | 1 | Attribut |
| C | 10d | Reservé au DOS |
| 16 | 2 | Heure de dernière écriture |
| 18 | 2 | Date de dernière écriture |
| 1A | 2 | N° de la 1ère Entrée en FAT |
| 1C | 2 | Taille du fichier (Nbre d'octets) |
| 1E | 2 | Libre |

Valeur de l'Attribut, ces valeurs sont additives.

01h = lecture seulement    02h = Caché → 03h = Fichier Système
0Bh = Nom de Volume    10h = Répertoire    20h = Archive
3Ff = N'importe quel fichier (c'est la somme des autres attributs)

## Calcul de l'adresse physique

(Les valeurs typiques indiquées sont pour les disquettes).

(Désolé, les noms de variables sont en anglais, c'est dans cette langue que j'ai écrit le programme).

En appelant :

| | | |
|---|---|---|
| NbSectorsTrack | le nombre de secteurs par piste | (8 or 9), |
| NbTracksSide | le nombre de pistes par face, | (40 or 80), |
| Temp | une valeur intermédiaire, | |
| SectorLogNb | le N° logique du secteur | (0...359/719/1 439), |
| SectorNb | le N° du secteur sur la piste, | (1...8 or 1...9), |
| HeadNb | le N° de la tête, | (0 or 1), |
| CylNb | le N° du cylindre | (0...39 or 0...79), |

On a :

SectorNb = (SectorLogNb mod NbSectorsTrack) + 1
Temp = SectorLogNb div NbSectorsTrack
HeadNb = Temp div NbHeads
CylNB = Sector LogNb div (NbHeads × NbSectorsTrack)

(NB : la formule donnée dans la Revue de l'Utilisateur de l'IBM PC n° 62 (Mai 1990) n'est valable qu'avec 1 seule tête).

## Utilisation de l'Interruption 13, fonction 2, lecture de secteurs

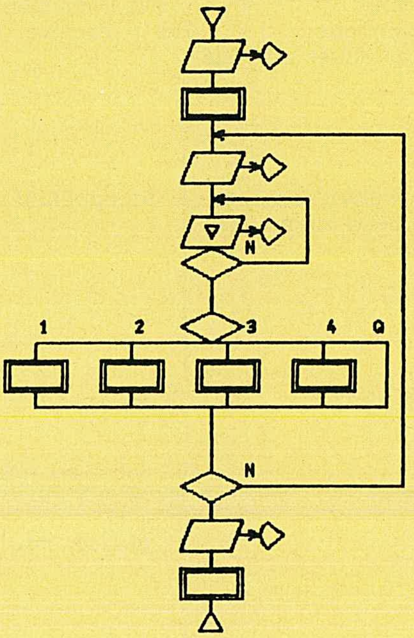
| | | | |
|---|---|---|---|
| AH : | 02 | AL : | Nbre de secteurs à lire |
| CH : | TrackNb | CL : | SectorNb |
| DH : | HeadNb | DL : | DriveNb (a : = 0 ; b : = 1,...) |
| ES : | Segment Address(Buffer) | BX : | Offset Address(Buffer) |

Renvoie un code d'erreur, éventuellement ≠ 0, en AH.

Le détail de ces codes est dans le listing.

Organigrammes principaux : (à l'attention de ceux qui ne sont pas coutumiers du Turbo Pascal).

Principal et Menu_1 :




Titre, auteur,...

Initialiser

Menu_1

Option
*Option in [1,2,3,4,Q] ?*

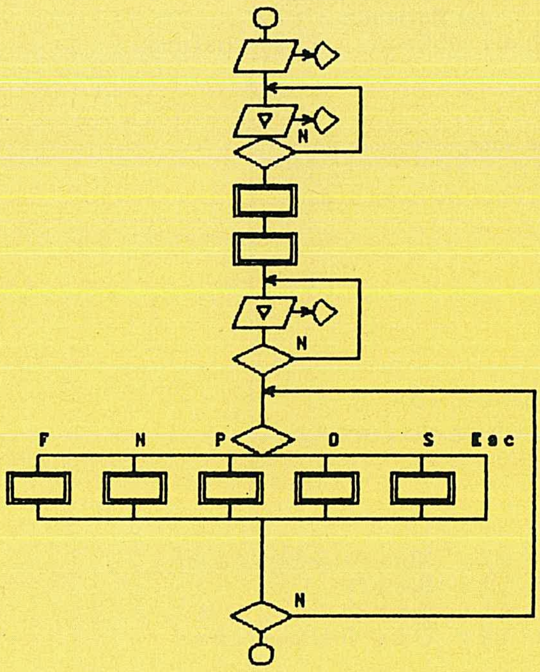
Cas
- 1 : Lister Root-Directory
- 2 : Lister Sub-directory
- 3 : Recupérer un fichier
- 4 : Réinitialiser
- Q : Quitter

*Option = Q ?*

Message de fin

Procédure de fin

Récupération et Menu_2 :




Menu

Entrer le 1er N° de Secteur
*Valide ?*

Lire le Secteur

Afficher le contenu

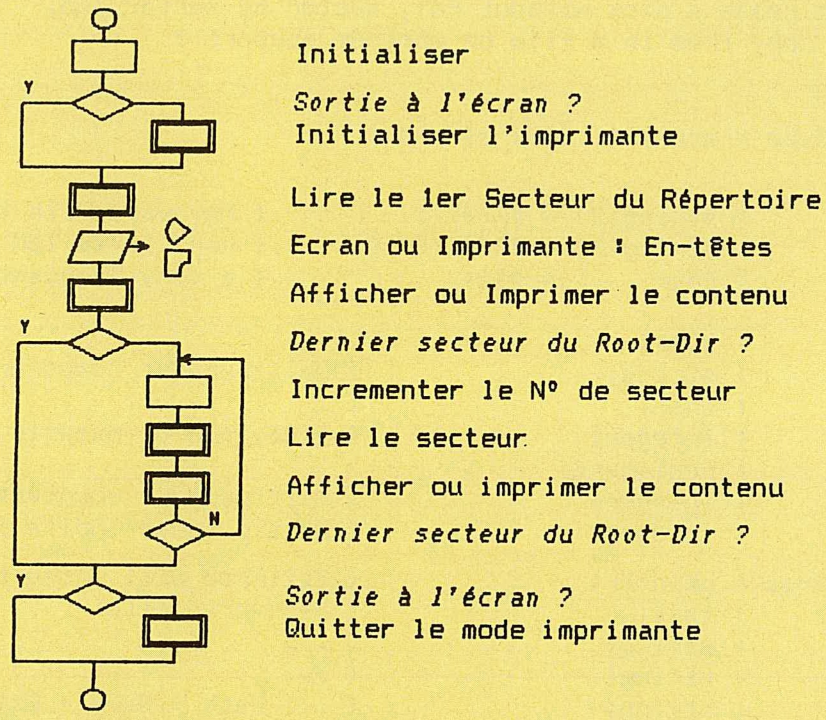
Option Menu_2

*Option in [F,N,P,O,S,Esc] ?*

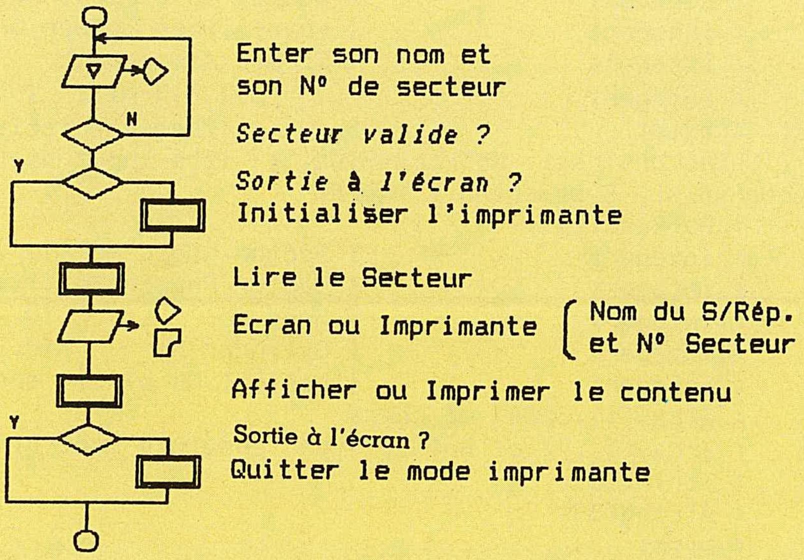
Case
- F : Get File Name
- N : Get Next Sector
- P : Get Previous Sector
- O : Get Other Sector
- S : Sauvegarde
- Esc : Retour au Menu_1

*Option = Esc ?*

Lister le Root Directory :

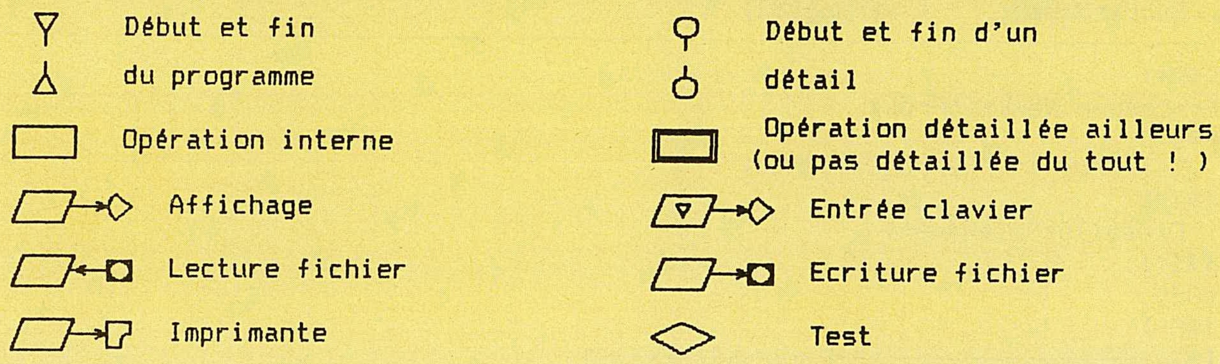



Lister un Sous-Répertoire :




------------------------------

Pour ceux qui n'auraient pas deviné voici le signification des symboles :



```
Program FAT_RBLD;                                { File Allocation Table Rebuilder }

{ This program reads a disk without FAT, sector by sector and,
  on request, copy them in a file on another support }

uses Crt,DOS;

type  BufSectType = array[1..512] of char;

const
  Elite            : string[2] = #27#77;         { May vary with the printer, }
  LeftMargin       : string[3] = #27#108#10;     { here it is IBM Proprinter, }
  FF               : char      = #12;            { a very frequent one !      }
var
  Option           : char;
  DumStr           : string;               { Multipurpose temporary string    }
  DumChar          : char;                 { .  .  .  .  .  .  char           }
  Error            : integer;              { BIOS, DOS or Numeric Value       }
  Regs             : registers;
  I, J             : integer;              { Multipurpose counters            }
  Index            : integer;              { Multipurpose Indice for arrays   }

  PreProgExitProc  : pointer;              { ExitProc will close the Outfile  }
  IntFileName      : text;                 { Recovered file                   }
  ExtFilePath      : string;               { ...                              }
  ExtFileName      : string;               { ...                              }
  OutFile          : string;               { ... Path + Name + Extension      }
  NameOk           : boolean;              { All char in '0'..'9','A'..'Z','_'}
  Saved            : boolean;              { To avoid saving twice in line    }
  Screen           : boolean;              { Output on Screen, not on Printer }
  LineNb           : integer;              { Nb of lines Screen or Printer    }
  Co,Li            : integer;              { Screen management                }
  InputUnit        : integer;                     { 0 = A: ;  1 = B:  }
  NbSectorsClust   : byte;                        {Characteristics of }
  NbBytesSector, NbSectorsTrack, NbTracksSide,    {the Input disk     }
  NbHeads, NbSectorsDisk, SystSectNb :  word;
  BufSect          : BufSectType;
  SectLogNb        : integer;              { Sector Logic Nb  (0..719 or 1439)}
  SectPhyNb        : integer;              { Sector Physic Nb (1..8 or 9)     }
  HeadNb           : integer;              { Head Nb          (0..1)          }
  CylNb            : integer;              { Cylinder Nb      (0..39 or 79    }
  FirstSector      : integer;              { Logical Nb of 1st sector to read }
  Root_Dir_Char    : array[1..3584] of char;
  Root_Dir_Sect    : array[1..7] of BufSectType  absolute Root_Dir_Char;
  FileName         : string[8];
  FileExt          : string[3];
  FileAttr         : byte;
  EntryNb          : word;
  FileSectNb       : word;
  FLength          : word;
  FileNbSectors    : word;
  SubDirName       : string[8];
{-------------------------------------------------------------------------}
{$F+}
Procedure NewExitProc;
begin
  ExitProc := PreProgExitProc;
{$I-}
  Close(IntFileName);
{$I+}
end;
{$F-}
{-------------------------------------}
```

```
Procedure Init_Printer;

begin
  Assign(Output,'LPT1');
  Rewrite(Output);
  Write(Elite + LeftMargin);
end;
{-----------------------------------}

Procedure Quit__Printer;
begin
  Write(FF); Close(Output)
  Assign(Output,'');
  Rewrite(Output);
end;
{--------------------------------------------------------------------}

Procedure Treat_Error;

begin
  Window(1,3,80,21);
  ClrScr;
  GotoXY(25,12);
  Write(Chr(7));
  Case Error of
    $01 : Write('Bad Command');
    $02 : Write('Wrong addressing');
    $03 : Write('Write protected disk');
    $04 : Write('Sector not found');
    $05 : Write('Driver init error');
    $07 : Write('Wrong parameters');
    $08, $09 : Write('Error DMA');
    $10 : Write('Error CRC');
    $11 : Write('Error ECC');
    $20 : Write('Disk Driver out');
    $40 : Write('Wrong positionning');
    $80 : Write('Time out');
    $FF : Write('Hard disk error');
    else Write('Unidentified error');
  end;
  GotoXY(70,24); Option := ReadKey;
  Halt(1);
end;
{-----------------------------------}

Procedure Initialize;

begin                                               { Get_Input_Output_Units }
  Window(1,3,80,25);  ClrScr;  Window(1,1,80,25);
  GotoXY(25,8); Write('Input disk unit (A/B)   ? ');
  Co := WhereX;
  Repeat
    GotoXY(Co,8);
    Option := UpCase(ReadKey);
    GotoXY(Co,8);
    Write(Option);
  until Option in ['A','B'];
  InputUnit := Ord(Option)-65;                          { Input Unit = 0 or 1 }
  With Regs do                          { Find the disk type and characteristics }
    begin
      repeat
        AX := $0201;                              { Function # 2 : read  1 sector }
        CX := $0001;                              { Track 0, Sector 1             }
        DH := $00;                                { Head  0                       }
```

```
          DL := InputUnit;                        { Unit Nb                    }
          ES := Seg(BufSect);
          BX := Ofs(BufSect);
          Intr($13,Regs);
          Error := AX div 256;
        until Error <> $80;                       { Reteat if 'Time out'       }
        If Error <> 0
          then Treat_Error
          else begin
            Window(25,10,80,25);
            NbHeads          := Ord(BufSect[27]);
            Writeln('Nb of Heads                  : ',NbHeads);
            NbSectorsClust := Ord(BufSect[14]);
            Writeln('Nb of Sectors per Cluster : ',NbSectorsClust);
            NbSectorsTrack := Ord(BufSect[25]);
            Writeln('Nb of Sectors per Track   : ',NbSectorsTrack);
            NbBytesSector   := 256*Ord(BufSect[13]) + Ord(BufSect[12]);
            Writeln('Nb of bytes per sector    : ',NbBytesSector);
            NbSectorsDisk   := 256*Ord(BufSect[21]) + Ord(BufSect[20]);
            Writeln('Nb of Sectors on disk     : ',NbSectorsDisk);
            NbTracksSide := (NbSectorsDisk div NbHeads) div NbSectorsTrack;
          end;
      end;
    Li := WhereY;
    GotoXY(1,Li+3);
    Write('Output Unit and Path : ');
    Co := WhereX;
    Repeat
      GotoXY(Co,Li+3);
      Readln(ExtFilePath); ClrEol;
    until (ExtFilePath[1] <> Option)
      AND (ExtFilePath[1] in ['a','A','b','B','c','C'])
      AND (ExtFilePath[2] = ':');
    OutFile := ExtFilePath;

    Window(1,3,80,25);  ClrScr;  Window(1,1,80,25);

    GotoXY(25,6); Write('Output of directories to');
    GotoXY(25,8); Write('Printer or Screen (P/S) ? ');          { Get_P_or_S }
    Co := WhereX;
    Repeat
      GotoXY(Co,8);
      Option := UpCase(ReadKey);
    until Option in ['P','S'];
    Write(Option);
    If Option = 'P'
      then Screen := False
      else Screen := True;

    If NOT Screen
      then begin
        GotoXY(25,10); Write('Prepare the printer ');
        Option := ReadKey;
      end;
    Window(1,3,80,25);  ClrScr;  Window(1,1,80,25);
end;
{---------------------------------------}

Procedure Read_Sector(SectLogNb : integer);

var
  Temp : integer;

begin
```

```
    SectPhyNb := (SectLogNb mod NbSectorsTrack) + 1;
    Temp      :=  SectLogNb div NbSectorsTrack;
    HeadNb    :=  Temp mod NbHeads;
    CylNb     :=  SectLogNb div (NbHeads*NbSectorsTrack);
    Window(1,1,80,25);
    GotoXY(1,23);
    Write('SectLogNb : ',SectLogNb);
    Write('     SectPhysNb : ',SectPhyNb);
    Write('     HeadNb : ',HeadNb);
    Write('     CylNb : ',CylNb,' ');

    With Regs do
      begin
        AX := $0201;
        CH := CylNb;              CL := SectPhyNb;
        DH := HeadNb;             DL := InputUnit;
        ES := Seg(BufSect);
        BX := Ofs(BufSect);
        Intr($13,Regs);
        Error := AX div 256;
        If Error <> 0
          then Treat_Error;
      end;
end;
{---------------------------------------}
Procedure Output_Sector_Dir(Index : integer);
begin
  Window(4,6,80,22);  ClrScr;
  I := 0;
  Repeat
    If (Ord(Root_Dir_Sect[Index,32*I + 12]) AND 8) = 8      { Attr : ...1.... }
      then begin
        For J := 1 to 11 do
          Write(Root_Dir_Sect[Index,J]);                      { Volume Id }
        Writeln('  Volume Id');
      end
      else begin
        For J := 1 to 8 do
          Write(Root_Dir_Sect[Index,32*I + J]);               { File Name }
        Write(' ');
        For J := 9 to 11 do
          Write(Root_Dir_Sect[Index,32*I + J]);               { Extension }

        FileAttr :=Ord(Root_Dir_Sect[Index,32*I + 12]);       { Attribute }
        If FileAttr >= 32
          then begin Write(' Archive  ');  FileAttr := FileAttr-32; end;
        If FileAttr >= 16
          then begin Write(' Directory');  FileAttr := FileAttr-16; end;
        If FileAttr >=  8
          then begin Write(' Volume Id');  FileAttr := FileAttr-8;  end;
        If FileAttr >=  4
          then begin Write(' SysFile  ');  FileAttr := FileAttr-4;  end;
        If FileAttr >=  2
          then begin Write(' Hidden   ');  FileAttr := FileAttr-2;  end;
        If FileAttr  =  1
          then begin Write('ReadOnly':10); FileAttr := FileAttr-1;  end;

        EntryNb := 256*Ord(Root_Dir_Sect[Index,32*I + 28])
                     + Ord(Root_Dir_Sect[Index,32*I + 27]);
        Write(EntryNb:6);

        If NbTracksSide = 40
          then SystSectNb := 12
```

```
          else SystSectNb := 14;
        FileSectNb := (EntryNb-2)*NbSectorsClust + SystSectNb;
        Write(FileSectNb:11);

        FLength := 256*Ord(Root_Dir_Sect[Index,32*I + 30])
                     + Ord(Root_Dir_Sect[Index,32*I + 29]);
        Write(FLength:12);

        If (Flength mod NbBytesSector <> 0)
          then FileNbSectors := 1
          else FileNbSectors := 0;
        FileNbSectors := FileNbSectors + Flength div NbBytesSector;
        Writeln(FileNbSectors:13);
      end;

    I := I + 1;
  until (Ord(Root_Dir_Sect[Index,32*I + 1]) = 0) OR (I = 16);
  Option := ReadKey;
end;
{-----------------------------------------------------------------------------}
Procedure Output_Root_Dir;

begin
  Window(4,4,80,22);  ClrScr;
  Write('Directory : ROOT      FAT Entry  Sector #  FileLength  ');
  Write('Nb of sectors');
  Writeln;
  Writeln;
  Window(1,1,80,25);
end;
{--------------------------------------}

Procedure List_Root_Directory;
begin
  For I := 1 to 3584 do
    Root_Dir_char[I] := ^@;
  If NbSectorsDisk = 720
    then FirstSector := 5
    else FirstSector := 7;
  If NOT Screen
    then Init_Printer;

  SectLogNb := FirstSector;
  Index     := 1;
  Read_Sector(SectLogNb);
  Root_Dir_Sect[Index] := BufSect;
  Output_Root_Dir;
  Output_Sector_Dir(Index);

  While (SectLogNb < FirstSector + 6) AND (BufSect[481] <> ^@) do
    begin
      Inc(SectLogNb);
      Inc(Index);
      Read_Sector(SectLogNb);
      Root_Dir_Sect[Index] := BufSect;
      Output_Sector_Dir(Index);
    end;
  If NOT Screen
    then Quit_Printer;
end;
{-----------------------------------------------------------------------------}
Procedure Output_Sub_Dir;

begin
  Window(1,3,80,25);  ClrScr;  Window(4,4,80,22);
```

```
    Write('Directory : ',SubDirName);
    DumStr := '';
    For I := 1 to (11 - Length(SubDirName)) do
     DumStr := DumStr + ' ';
    Write(DumStr,'FAT Entry  Sector #  FileLength  Nb of sectors');
    Writeln;
    Writeln;
    Window(1,1,80,25);
  end;
  {-----------------------------------}

  Procedure List_Sub_Directory;

  begin
    Window(1,3,80,21);  ClrScr;
    Writeln;
    Write('Directory Name ');
    Readln(SubDirName);
    GotoXY(41,1);
    Write('First Sector Nb : ');

    Repeat
      GotoXY(59,1);
      Readln(DumStr);
      Val(DumStr,FirstSector,Error);
    until (Error = 0) AND (FirstSector >= 0) AND (FirstSector < NbSectorsDisk);

    For I := 1 to 512 do
      BufSect[I] := ^@;
    If NOT Screen
      then Init_Printer;

    SectLogNb := FirstSector;
    Read_Sector(SectLogNb);
    Root_Dir_Sect[1] := BufSect;
    Output_Sub_Dir;
    Output_Sector_Dir(1);

    If NOT Screen
      then Quit_Printer;
  end;
  {----------------------------------------------------------------------}

  Procedure Window_1;

  begin Window(1,3,80,21) end;
  {----------------}

  Procedure Window_2;

  begin Window(1,25,80,25) end;
  {-----------------------------------}

  Procedure Menu_2;

  begin
    Window_2;
      HighVideo; Write('F'); LowVideo;  Write('ile_name : ',ExtFileName);
      GotoXY(23,1);
      HighVideo; Write('S'); LowVideo; Write('ave  ');
      HighVideo; Write('N'); LowVideo; Write('ext_sector  ');
      HighVideo; Write('P'); LowVideo; Write('revious_sector  ');
      HighVideo; Write('O'); LowVideo; Write('ther_sector  ');
      HighVideo; Write('<Esc>'); LowVideo;
  end;
  {-----------------------------------}
```

```
Procedure Get_File_Name;

begin
  GotoXY(13,1);
  Write('........');
  Repeat
    GotoXY(13,1);
    Readln(ExtFileName);
    ClrEol;
    NameOk := True;
    For I := 1 to Length(ExtFileName) do
      begin
        ExtFileName[I] := UpCase(ExtFileName[I]);
        If NOT (ExtFileName[I] in ['0'..'9','A'..'Z','_'])
          then NameOk := False;
      end;
    Menu_2;
    GotoXY(13,1);
    Write(ExtFileName);
  until NameOk AND (Length(ExtFileName) > 0);
  OutFile := OutFile + ExtFileName + '.RCV';
end;
{-------------------------------------}

Procedure Display_Data_Sector;
begin
  Window_1;
  GotoXY(1,19); (* 1 *)
  I := 0;
  While I < 512 do
    begin
      Inc(I);
      If (BufSect[I] = '\') AND (BufSect[I+1] = ',')
        then begin
          Writeln;
          I := I + 2;
        end
        else If (Ord(BufSect[I]) > 31)
               then Write(BufSect[I]);
    end;
  Writeln;
  Writeln;
end;
{-------------------------------------}
Procedure Get_Next_Sector;
begin
  Window_1;
  GotoXY(1,21);
  If (SectLogNb < NbSectorsDisk - 1)
    then begin
      SectLogNb := SectLogNb + 1;
      Read_Sector(SectLogNb);
      Display_Data_Sector;
      Saved := False;
    end
    else begin

      GotoXY(1,21);
      Writeln;
      Write('This was the last sector', #7);
    end;
end;
```

```
{--------------------------------------}

Procedure Get_Previous_Sector;

begin
  Window_1;
  GotoXY(1,21);
  If (SectLogNb > 0 )
    then begin
      SectLogNb := SectLogNb -1;
      Read_Sector(SectLogNb);
      Display_Data_Sector;
      Saved := False;
    end
    else begin
      GotoXY(1,21);
      Writeln;
      Write('This was the first sector', ^G);
    end;
end;
{--------------------------------------}

Procedure Get_Other_Sector;
begin
  Window_1;
  Repeat
    GotoXY(1,19);
    Write('Sector Logical Nb : ');
    Readln(DumStr);
    Val(DumStr,SectLogNb,Error);
  until (Error = 0) AND (SectLogNb >= 0) AND (SectLogNb < NbSectorsDisk);
  Read_Sector(SectLogNb);
  Display_Data_Sector;
  Saved := False;
end;
{--------------------------------------}
Procedure Save_Sector;
begin
  If Saved then exit;
  Saved := True;
  If OutFile = ExtFilePath
    then Get_File_Name;
{$I-}
  Assign(IntFileName,OutFile);
  Append(IntFileName);
{$I+}
  Error := IOResult;
  If Error <> 0
    then begin
      Window_1;
      Case error of
        2 : Rewrite(IntFileName);
        3 : Writeln('DOS error : Path not found',^G);
        else begin
          Writeln('DOS error other than ''Path not found''',^G);
          Halt;
        end;
      end;
    end;
  If Error in [0,2]
    then begin
```

```
        I := 0;
        Repeat
          Inc(I);
          Write(IntFileName,BufSect[I]);
        until (I = 512) OR (BufSect[I] = ^Z);
        Close(IntFileName);
        GotoXY(21,1);
        Writeln('Saved');
      end;
    Menu_2;
  end;
{---------------------------------------}

Procedure Recover;

begin
  Window(1,3,80,25);  ClrScr;  Window(1,1,80,25);
  GotoXY(1,24);
  Write('----------------------------------------');
  Write('----------------------------------------');
  GotoXY(1,22);
  Write('----------------------------------------');
  Write('----------------------------------------');
  ExtFileName := '........';
  OutFile := ExtFilePath;
  Menu_2;
  Window_1;
    Repeat
      GotoXY(1,19);
      Write('Sector Logical Nb : ');
      Readln(DumStr);
      Val(DumStr,SectLogNb,Error);
    until (Error = 0) AND (SectLogNb >= 0) AND (SectLogNb < NbSectorsDisk);
    Read_Sector(SectLogNb);
    Display_Data_Sector;

  Repeat
    Window_2;
    Repeat
      GotoXY(79,1);
      Option := UpCase(ReadKey);
    until Option in [#27,'F','N'..'P','S'];
    Case Option of
      'F' : Get_File_Name;
      'N' : Get_Next_Sector;
      'P' : Get_Previous_Sector;
      'O' : Get_Other_Sector;
      'S' : Save_Sector;
    end;
  until Option = #27;
end;
{-----------------------------------------------------------------------------}
```

# Si vous êtes déjà un utilisateur de CLIPPER de Nantucket, vous avez besoin du

## Générateur de programme ATELIER

Il fait pour vous :

- ☐ la génération du menu principal,
- ☐ la génération de sous-menus,
- ☐ la génération des programmes de saisie,
- ☐ la génération des programmes d'impression,
- ☐ la gestion de vos sources,
- ☐ le lancement du compilateur...

## ATELIER

- ☐ incorpore votre copyright et une calculette dans vos programmes
- ☐ génère le code fastidieux (90 %)
- ☐ et vous laisse faire la partie intéressante de la programmation.

## ATELIER

☐ est livré avec les sources commentées, ainsi qu'une documentation imprimable.

La version compilée n'est pas livrée sur la disquette car ATELIER est destiné aux utilisateurs de CLIPPER et nécessite ce compilateur.

*Les commentaires sur ce programme ont été publiés dans La Revue de l'Utilisateur de l'IBM PC (numéro 59, Janvier-Février 1990), et la suite de la série comportera 6 articles à compter du numéro 61, Avril 1990.*

## ATELIER

est disponible auprès de La Revue de l'Utilsateur de l'IBM PC au prix de 450 F, sur disquette 5''1/4 ou 3''1/2.

---

COUPON

à adresser à La Revue de l'Utilisateur de l'IBM PC

2, rue d'Amsterdam

75009 Paris

Tél. : 47.64.07.57. - 42.67.93.80

Nom ........................................ Prénom ........................................

Adresse ..............................................................................

..............................................................................

Code Postal ........................................ Ville ........................................

Téléphone ..............................................................................

Veuillez m'envoyer ....... logiciel (s) ATELIER au prix unitaire de 450 F (TTC).

Veuillez trouver ci-joint un chèque de ......... F en règlement à Softin Communications.

```
Procedure Menu_1;

begin
  Window(1,3,80,25); ClrScr;  Window(1,1,80,25);
  GotoXY(23,5);  Write('List the files : Root Directory  : 1');
  GotoXY(40,7);  Write('Other Directory : 2');
  GotoXY(23,10); Write('Recover a file                   : 3');
  GotoXY(23,12); Write('Reinitialize                     : 4');
  GotoXY(23,15); Write('Quit                             : Q');
  GotoXY(44,18); Write('Your Choice : ');
  Co := WhereX;
  Option := ^@;
  Repeat
    GotoXY(Co,18);
    Option := ReadKey;
    Write(Option);                  (* 1 *)
  until Option in ['1'..'4','Q','q'];

  Case Option of
    '1' : List_Root_Directory;
    '2' : List_Sub_Directory;
    '3' : Recover;
    '4' : Initialize;
  end;
end;
{-----------------------------------------------------------------------------}

begin                                   { Main body }

  PreProgExitProc := ExitProc;
  ExitProc := @NewExitProc;
  ClrScr;
  GotoXY(17,1); Writeln('FAT Rebuilder V1-0, by J. ROULLIER - 17th July 1991');
  Write('------------------------------------------');
  Write('------------------------------------------');
  Initialize;
  Repeat
    Menu_1
  until (UpCase(Option) = 'Q');
  Window(1,1,80,25);  ClrScr;
  GotoXY(36,8);  Write('Good bye');
  GotoXY(22,12); Write('I wish you don''t have to use me again');
  GotoXY(1,24);
end.
```

------------------------------

# Topspeed

## *Le spécialiste de la compilation*

**Générateur optimisé** Haute qualité du code généré, optimisation de l'édition de liens (toute variable ou fonction non utilisée n'est pas incluse dans l'exécutable).

**Plus de limite à 640 Ko** Selon que vous choisissez de travailler en mode réel ou mode protégé, la mémoire disponible pour vos programmes peut atteindre de 16 MO à 512 MO. TopSpeed gère automatiquement les overlays pour le code et les données, les librairies dynamiques (DLL) chargées et déchargées à la demande et utilise même le disque en mémoire virtuelle de vos programmes.

**Multi–tâches** Jusqu'à 32 tâches concurrentes même sous DOS.

**Code source disponible** Toutes les librairies peuvent être fournies en code source.

**Portabilité** Génère du code pour DOS, OS/2, et Windows 3.

**Multi langages** Choisissez le langage le mieux adapté au traitement que vous allez réaliser; TopSpeed permet de construire une application à partir de modules écrits dans des langages differents (C, C++, Pascal, Modula–2). Utiliser n'importe quelle librairie à partir de n'importe quel langage. Développez dans des langages différents en utilisant le même éditeur et le même environnement.

**Environnement *TopSpeed*** Commun à tous les langages, éditeur 10 fenêtres d'un Mo chacune, gestionnaire de projet, **aide hypertexte** (langage et librairie), debugger, contrôle de syntaxe, TSR pour calculatrice, **table Ascii**, scan–code, etc.
DOS ou OS/2 **590,00 F**

***TopSpeed C*** Certifié ANSI pour portabilité maximum, génération automatique des prototypes, compatible Turbo C, Microsoft C 5.1 et Quick C, génère du code compact de haute qualité pour DOS, OS/2 et Windows 3, contrôle des anomalies à l'exécution (pointeur nul, dépassement de bornes, etc).
DOS ou OS/2 **590,00 F**

***TopSpeed* C++** Conforme AT&T 2.1, programmation orientée objet. Editeur de liens **Smart Method** élimine automatiquement les classes et méthodes non référencées dans le code.
DOS ou OS/2 **590,00 F**

***TopSpeed* Pascal** Conforme ISO 7185 Niveau 1, extensions orientés objets, array et string dynamiques, convertisseur Turbo Pascal, compilation séparée des unités, etc.
DOS ou OS/2 **590,00 F**

***TopSpeed* Modula 2** Successeur de Pascal, langage très structuré et orienté objets, conversion de type automatique.
DOS ou OS/2 **590,00 F**

***TopSpeed* TechKit** DLL même sous DOS, Debugger Post–Mortem (permet d'analyser les causes d'un «plantage», Profiler (permet d'optimiser la vitesse d'exécution de votre programme en détectant les sections les plus coûteuses en temps), assembleur ***TopSpeed***, compresseur d'EXE, désassembleur, librairies et compilateur de ressources pour Windows 3.
DOS ou OS/2 **590,00 F**

**Nouveau**

***TopSpeed* DOS Extender**

Fonctionnement en mode protégé sur processeur 80286 et **au delà**. Jusqu'à 512 MO de programmes et données chargés et déchargés automatiquement à la demande de l'application. Supporte XMS, DPMI, VCPI, API OS/2.
DOS ou OS/2 **1180,00 F**

**Offre Promotionnel**

| | ☐ **Mode Réel** | | ☐ **Mode Protégé** |
|---|---|---|---|
| Environnement | 590,00 | Environnement | 590,00 |
| + 1 compilateur | 590,00 | + 1 compilateur | 590,00 |
| | | + DOS Extender | 1180,00 |
| | ~~1180,00 HT~~ | | ~~2360,00 HT~~ |
| | 990,00 HT | | 1990,00 HT |
| **+ TVA + frais d'envoi** | **1249,14 TTC** | | **2435,14 TTC** |

**Bon de commande**

Veuillez trouver ci–joint mon règlement par chèque pour l'offre promotionnelle de ☐ MODE Réel ☐ MODE Protégé – en version anglaise à l'ordre de :

**BIONIX** – 32, rue Molitor – 75016 Paris
Tél : 40 71 09 36 Fax : 46 51 24 26

TopSpeed® C++ TopSpeed® Modula-2 TopSpeed® Pascal TopSpeed® C

LE CONTROLEUR DU logiCIEL
ARGOS PRO
UN NOUVEAU CONCEPT DANS LE DOMAINE DE LA PROTECTION DE LOGICIELS
Notre expérience dans le domaine de la protection, notre pouvoir d'innovation, nous permettent de vous proposer le seul produit Français utilisant des composants du type ASIC, pour une clé gigogne.
Avantages: Circuit intégré haute densité exclusivement fabriqué pour Electryon, permettant une clé de dimension réduite, une fiabilité au fonctionnement, une transparence parfaite et une protection absolue.
Seule clé dotée d'un numéro privé, elle permet d'identifier un PC par rapport à un autre.
Code éditeur confidentiel sur 48 bits dont 16 programmables par vous-même · 32 à plus de 200 registres de 16 bits disponibles en lecture et écriture.
ENCORE PLUS PERFORMANTE
Protection de 20 sociétés parmi 4000. Chacune de ces sociétés peut protéger plusieurs logiciels
Activation et désactivation de la clé par programme.
Garantie : de 1 à 3 ans selon modèle.
DEMANDEZ NOTRE DOCUMENTATION
ELECTRYON
53, rue Corot · La Rochette · 77000 MELUN · FRANCE.
Tél: 33(1)64 39 13 33 · Téléfax: 33(1)64 39 17 81
CREATIVE AUXERRE
Argos
Argos
Argos
18B7 · 3A2C

**N° 1 AUX ETATS-UNIS**
EN VERSION FRANCAISE

## Réalisez vous-même vos applications personnalisées sans taper une seule ligne de programme :

Créez des applications relationnelles de qualité et distribuez-les librement. Base de données, multi-fichiers indexés, écran multifenêtres, menus, tableaux déroulants, masques de saisie, édition de toute nature, champs calculés, conditionnels, totaux, etc...

**C'est facile ! C'est étonnant !**

Matériel requis : micro-ordinateur compatible PC et système d'exploitation MS-DOS, 512 K RAM, 2 MO sur disque.

Spécifications techniques : Multifichiers, 252 fichiers par application, 251 clés par fichier, 255 champs par enregistrement. Longueur enregistrement jusqu'à 64 Ko.

Taille des fichiers : illimitée.

Mode automatique : crée une application totale en quelques secondes (menus, tableaux déroulants, masques de saisie, états) à partir de la simple définition du fichier.

Aide à l'utilisation : intégrée (plus de 400 écrans).

Conversion de fichiers : import/export avec fichiers dBase, DIF, ASCII délimité.

**EN CADEAU**

Huit applications types, déjà développées, utilisables telles quelles ou modifiables. (Inventaire, gestion de club, mailing, facturation, répertoire téléphonique, suivi commercial, gestion scolaire).

BIONIX SOFTWARE
32, rue Molitor 75016 Paris
Tél : (33) (1) 40 71 09 36
Fax : (33) (1) 46 51 24 26

**Pour la Suisse :**
**IHS Distribution S.A.**
En Budron A 16
1052 Le Mont / Lausanne
Tél : (021) 652 34 36
Fax : (021) 652 34 52

**Pour la Belgique :**
**Nasa Informatique S.A.**
Rue Godefroid, 30 5000 Namur
Tél : (081) 22 40 77
Fax : (081) 23 19 35

# LE PRIX DE LA PERFORMANCE: 990,00 F HT

PRIX DE LANCEMENT

PRIX TARIF : 2390,00 F HT.

EDITION SPECIALE

PHOTO COMSTOCK

**CLARION Personnel**

LE GENERATEUR D'APPLICATION UNIVERSEL

BON DE COMMANDE à remplir et à retouner à BIONIX.

Je désire recevoir CLARION Personnel au prix de lancement de 990, 00 Francs HT plus la TVA et ma participation aux frais d'envoi soit 1249,14 francs TTC.
Veuillez trouver ci-joint mon chèque de réglement à l'ordre de BIONIX.

❏ Disquettes 3 1/2 pouces Signature
❏ Disquette 5 1/4 pouces

Nom ______ Société ______
Fonction ______ Tél ______
Adresse ______
______
Code Postal |_|_|_|_|_| Ville ______

# MARLO le réflexe micro

## ACER 1120/SX

80386SX-20 — 2 Mo RAM — 1 disque dur 40 Mo — VGA couleurs — Garantie 1 an sur site
**Prix MARLO : 13 500 F**

## SHARP PC6661

80386SX-20 MHz — 2 Mo RAM — 1 lecteur 3''1/2 — Disque dur 60 Mo — Écran LCD VGA — DOS — Poids : 2,7 kg
**Prix MARLO : 17 990 F**

## COMPAQ LTE 386S/20

80386SX-20 MHz — 2 Mo RAM — 1 lecteur 3''1/2 — Disque 30 Mo — Écran LCD VGA — Poids : 3,4 kg
**Prix MARLO : 21 990 F**
Version 60 Mo : 23 790 F

## TOSHIBA T1000LE

80C86 à 9,54 MHz — 1 Mo RAM — Disque dur 20 Mo — LCD 640×400 — Poids : 3 kg — DOS
**Prix MARLO : 10 500 F**

## TOSHIBA T2000SXe

80386SX-20 — 2 Mo RAM — Disque dur 40 Mo — LCD VGA — Poids : 2,5 kg — DOS
**Prix MARLO : 21 990 F**
Version 60 Mo : 23 490 F

## COMMODORE DT286-12

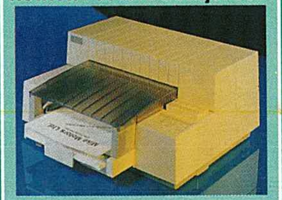
80286-12 — 1 Mo RAM — 1 lecteur 3''1/2 — Disque dur 40 Mo — Écran VGA couleurs 14'' — MSDOS — WORKS 2 + GEOWORKS
**Prix MARLO : 6 300 F**

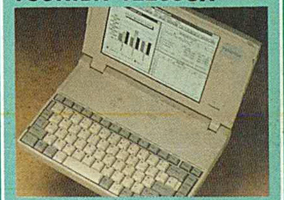
## COMMODORE 286LT

80286-12 — 1 Mo de RAM — 1 lecteur 3''1/2 — Disque dur 20 Mo — Écran LCD VGA — DOS — Housse-Works 2 — Autonome
**Prix MARLO : 9 900 F**

## SHARP 8501

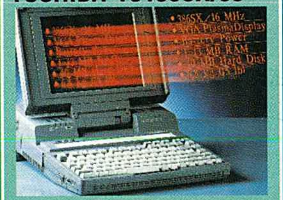
80386DX-20 — 2 Mo RAM — 1 lecteur 3''1/2 — Disque dur 100 Mo — Écran LCD VGA couleurs — DOS
**Prix MARLO : 43 800 F**

## Hewlett Packard Deskjet 500

Imprimante à jet d'encre — 2 à 3 pages/minute — Garantie 3 ans dont 1 an sur site.
**Prix MARLO : 3 400 F**

## TOSHIBA T2200SX

80386SX à 20 MHz — 2 Mo RAM — Disque 40 Mo — Écran LCD VGA — DOS — Poids : 2,5 kg — Autonomie : 4 heures.
**Prix MARLO : 23 490 F**
Version 60 Mo : 24 490 F

## VICTOR V386MX/40

80386-16 — 2 Mo RAM — Disque dur 52 Mo — Écran VGA couleurs — DOS — Windows
**Prix MARLO : 13 500 F**

## TOSHIBA T3100SX/80

80386SX-16 — 1 Mo RAM — Disque dur 80 Mo — Plasma VGA — Poids : 6,8 kg — DOS
**Prix MARLO : 20 990 F**

## COMPAQ 386/25e

80386-25 — 4 Mo RAM — Disque dur de 60 Mo - Écran VGA couleurs
**Prix MARLO : 29 700 F**

## TOSHIBA T3200SXC

80386SX-20 — 1 Mo RAM — Disque dur de 120 Mo — LCD VGA couleurs — Poids : 7,9 kg — DOS
**Prix MARLO : 43 990 F**

## TOSHIBA T5200

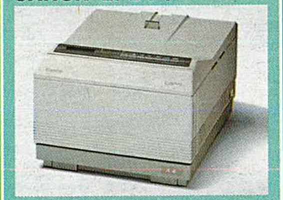
80386-20 — 2 Mo RAM — Disque dur 100 Mo — Plasma VGA — Poids : 8,5 kg — DOS
**Prix MARLO : 26 990 F**
Version 200 Mo : 30 990 F

## STAR LS8 III

8 ppm — 1 Mo RAM — Garantie 1 an sur site
**Prix MARLO : 10 990 F**
Version PostScript : 16 500 F

## STAR LS4

4 ppm — 1 Mo RAM — Garantie 1 an sur site
**Prix MARLO : 6 990 F**
Version PostScript : 9 790 F

## CANON LPB4+

4 pages minute — 512 Ko RAM — 300 Dpi — Cassette 50 feuilles — Garantie 1 an sur site
**Prix MARLO : 5 990 F**

## Hewlett Packard Série III

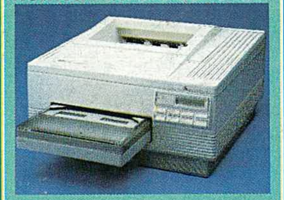
8 ppm — 1 Mo de RAM — Garantie 1 an sur site
**Prix MARLO : 11 990 F**
Version PostScript : 14 790 F

## CHICONY NB

80386SX-20 — 1 Mo RAM — Disque dur 40 Mo — LCD VGA — Poids : 3,4 kg — DOS — Garantie 3 ans
**Prix MARLO : 12 500 F**
Version 60 Mo : 14 500 F

## TOSHIBA T2000

80286 à 12 MHz — 1 Mo RAM — Disque 20 Mo — Écran LCD VGA — DOS — Poids : 3,1 kg
**Prix MARLO : 14 990 F**
Version 40 Mo : 15 890 F

## Hewlett Packard Série III P

4 pages minute — 1 Mo RAM — 300 Dpi — Garantie 1 an sur site.
**Prix MARLO : 7 990 F**
Version PostScript : 11 195 F

## SHARP PC6240

80C286 à 12 MHz — 1 Mo RAM — LCD VGA — Disque 40 Mo — Poids : 2,2 kg — DOS
**Prix MARLO : 12 500 F**

Financement Crédit Universel CB

MARLO

13, boulevard Ch.-de-Gaulle - 92700 COLOMBES

Tél. : (1) 47 81 20 57
Fax : (1) 47 80 59 63
OUVERT DU LUNDI AU SAMEDI

*Tous nos prix s'entendent hors taxes T.V.A. 18,6 % en sus. Prix modifiables sans préavis.*